Code CC/Ke

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

RESOURCE USAGE FOR ADAPTIVE C4I MODELS IN A HETEROGENEOUS COMPUTING ENVIRONMENT

by

N. Wayne Porter

June 1999

Thesis Advisor:	Debra Hensgen
Co-Advisor:	William G. Kemple

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 8

19990802 155

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE: RESOURCE USAGE FOR ADAPTIVE C4I MODELS IN A HETEROGENEOUS COMPUTING ENVIRONMENT		5. FUNDING NUMBERS
6. AUTHOR(S) Porter, N. Wayne		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT: Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE:

13. ABSTRACT *(maximum 200 words)*

The goal of the Management System for Heterogeneous Networks (MSHN) is to provide a resource management system (RMS) to enable adaptive applications to use multiple sets of shared resources while accounting for dynamically changing priorities and environments. This RMS must be capable of providing each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences) in a heterogeneous computing environment in which many processes are competing for shared resources.

Applying this RMS technology to C4I modeling and simulation applications would enable on-scene Commanders to simulate complex elements of the decision process in order to optimize the use of forces and materiel.

The objective of this thesis was to transparently intercept operating system calls made by a robust, C4I modeling application, the Extended Air Defense Simulation (EADSIM), in order to weigh the resources required against the confidence level of the outcomes obtained. Specifically, the goal was to determine resource usage required to run the application using both Monte Carlo simulation and deterministic simulation. MSHN needs this type of information to determine which version of an application to execute, in order to provide the best Quality of Service, while meeting operational deadlines.

14. SUBJECT TERMS C4I, Wrapper, Resource Management System, Intercept System Calls, Distributed System, Modeling and Simulation, Warfighter, Client Library, MSHN, Heterogeneous Computing, Quality of Service, Stochastic, Deterministic, Monte Carlo			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

RESOURCE USAGE FOR ADAPTIVE C4I MODELS IN A HETEROGENEOUS COMPUTING ENVIRONMENT

N. Wayne Porter
Lieutenant Commander, United States Navy Reserve
B.A., University of Southern California, 1974

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1999

Author: [illegible]
N. Wayne Porter

Approved by: [illegible]
Debra Hensgen, Thesis Advisor

[illegible]
William G. Kemple, Thesis Co-Advisor

Dan C Boger
Dan Boger, Chairman
Computer Science Department

ABSTRACT

The goal of the Management System for Heterogeneous Networks (MSHN) is to provide a resource management system (RMS) to enable adaptive applications to use multiple sets of shared resources while accounting for dynamically changing priorities and environments. This RMS must be capable of providing each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences) in a heterogeneous computing environment in which many processes are competing for shared resources.

Applying this RMS technology to C4I modeling and simulation applications would enable on-scene Commanders to simulate complex elements of the decision process in order to optimize the use of forces and materiel.

The objective of this thesis is to transparently intercept operating system calls made by a robust, C4I modeling application, the Extended Air Defense Simulation (EADSIM), in order to weigh the resources required against the confidence level of the outcomes obtained. Specifically, the goal is to determine resource usage required to run the application using both Monte Carlo simulation and deterministic simulation. MSHN needs this type of information to determine which version of an application to execute, in order to provide the best Quality of Service, while meeting operational deadlines.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

The Defense Advanced Research Projects Agency (DARPA)-sponsored Management System for Heterogeneous Networks (MSHN) project is a sub-component of the DARPA QUORUM program. The goal of this project, as well as the program at large, is to provide a resource management system (RMS) to enable military computer applications to use multiple sets of shared resources while accounting for dynamically changing priorities and environments. MSHN's RMS must be capable of providing each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences) in a heterogeneous computing environment in which many processes are competing for shared resources. Rather than establishing this RMS as stand-alone software, the MSHN architecture is designed as an integrated system, integrated with and incorporating a variety of distributed system tools to reap the maximum benefits from available resources.

In a military environment such an integrated system might mean offering the Commander the opportunity to select the most appropriate application, or version of an application, capable of executing within a specified time, at the proper security level, in order to deliver the best achievable answer within his stated time constraints. Applying this technology to robust C4I modeling and simulation applications would enable on-scene Commanders or mission planners to simulate complex elements of the decision process in order to optimize the use of forces and materiel.

The need for robust C4I modeling and simulation in support of the tactical commander, the "Warfighter," has been established in numerous Joint Warrior Interoperability Demonstrations, Fleet Battle Experiments, and Joint and service-specific exercises. However, to make the use of such models practical in a heterogeneous computing environment, the resource management concepts addressed by MSHN are needed.

Key to the implementation of MSHN is the requirement for adaptive and adaptation aware applications. Adaptive applications exist in different versions capable of producing like results (though possibly offering varying degrees of QoS). MSHN would monitor the use of such adaptive applications and would be able to terminate one

version and start another, possibly from the beginning, if it perceived the user's QoS requirements were not being met by the currently executing version. In a tactical environment, this means that the transition to improved QoS recommended by the MSHN RMS, if accepted by the decision-maker, would transparently enhance his or her mission effectiveness while remaining within given time constraints.

This thesis presents the methodology of intercepting, or wrapping, system calls made by the Extended Air Defense Simulation (EADSIM), a robust C4I, air and missile warfare modeling application, in order to determine the resources required to execute the program on a stand-alone workstation. Having demonstrated the ability to measure the application's resource usage without requiring access to source code, an experiment is described in which the resource usage is measured running the application in both Monte Carlo simulations and deterministic simulations. The outcomes obtained from running EADSIM in both deterministic and stochastic simulations are then weighed against each other. The overhead associated with the MSHN wrapper, a modified C library used to intercept system calls, is also measured and possible causes of this overhead are discussed.

The research conducted for this thesis led to the realization that the MSHN wrapper needs to be expanded to collect finer grained information. Specifically, `send( )`, `sendto( )`, `sendmsg( )`, `recv( )`, `recvfrom( )`, `recvmsg( )`, `select( )`, and `listen( )` system calls may need to be wrapped. Additionally, more information may be required from the current wrapper. Finally, the Binomial distribution was used to help evaluate the trade-off between the fidelity of results from deterministic simulations and stochastic simulations. It was concluded that counter-missile events cannot be assumed to be independent in order to develop a measure of effectiveness to compare the deterministic version of EADSIM with the stochastic version. However, it was shown that the deterministic version of EADSIM offers valuable information while requiring approximately 1/20 the compute resources required by the stochastic version.

ACKNOWLEDGEMENTS

I would like to sincerely thank Kate, Keegan, Ryan, and Shannon for their patience, sacrifice, understanding, and support. I would also like to thank my father for the dream, my mother and my sister for helping to make it possible. Professionally, I would like to thank my advisors, Professors Hensgen and Kemple, and two brilliant classmates, MAJ Matt Schnaidt, USA and MAJ Tom Cook, USA.

I. INTRODUCTION

This thesis investigates the need for adaptive[1], combat modeling; specifically as it relates to Command, Control, Communications, Computers and Intelligence (C4I). Further, it presents the results and analysis of resource measurement experiments run using the Extended Air Defense Simulation (EADSIM) program. The purpose of these experiments is to obtain data to facilitate the comparison of distributions of resources required to run the EADSIM program stochastically and the resources required to run it deterministically, weighing confidence levels of the results against the resources required.

A. C4I MODELING AND SIMULATION IN A DISTRIBUTED ENVIRONMENT

> Future military missions will be highly diverse, including operations other than war (OOTW), often undertaken by joint, combined, or coalition forces. However, existing command support tools are not flexible enough to aid commanders in planning for such diverse and dynamic missions. [DESI98]

The need for sophisticated C4I modeling and simulation[2] applications to support strategic planning has been accepted since the early part of the twentieth century when Frederick Lanchester developed his linear and square law equations during the First World War. However, the advent of "network-centric warfare," the concept of metacomputing, advances in communication technologies (offering dramatic increases in data transfer rates), and the ability to take better advantage of distributed systems have presented the opportunity to apply these same models and simulations in an operational environment for use by the Warfighter in tactical planning.

---

[1] Adaptive computer applications are those applications that, in order to support varying Quality of Service requirements, exist in different versions capable of producing like results. Adaptive applications should be idempotent, allowing them to be re-started in another version without corrupting any resources.

[2] Use of the terms "simulation" and "modeling" in this paper are in accordance with military common usage. The Department of Defense defines a "model" as: "A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process." "Simulation" is defined as: "A method of implementing a model over time." [MSMP95]

The implementation of the Global Command and Control System (GCCS) in 1996 heralded a major technological advance from its predecessor, the World Wide Military Command and Control System. GCCS software is a suite of applications intended to operate over a global network in support of the command and control mission of the United States and her coalition partners. As such it would seem to be a logical candidate as a venue for C4I modeling and simulation packages. The utility of having robust planning and decision tools available to the on-scene commander and his staff is obvious if (and only if) such tools can deliver data within very limited time constraints and with sufficient fidelity without robbing resources needed for the execution of such plans.

Department of Defense Joint Publication 1 defines command and control as follows:

> The exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission. Command and Control functions are performed through an arrangement for personnel, equipment, communications, facilities, and procedures which are employed by a commander in planning, directing, coordinating and controlling forces and operations in the accomplishment of the mission. [JOIN91]

This definition was used by Frank Snyder [SNYD93] to break out three fundamental elements of command and control: 1) the command function; 2) the command and control process; and, 3) command, control and communications (C3) systems (C3 systems have more recently been expanded to include computers and intelligence, resulting in the popular use of "C4I systems" vice simply "C3 systems"). In addition to providing us a three sided prism through which we might diffuse the complicated spectrum of command and control, Snyder identified two essential variables that impinge on every aspect of command and control, particularly during combat: time and uncertainty. It is in an attempt to aid the decision-makers (the commanders) in successfully managing, or balancing, these two variables and their effects that modeling and simulation provide ready tools. This desire for a "balanced" response underscores the need for a means to manage computing resources (resource management) so as to ensure that high priority applications will execute with acceptable timeliness and level of confidence.

The Lawson-Moose C4I Cycle (Figure 1. 1) provides a graphic depiction of the command and control decision process, commonly referred to as the "OODA" Loop, wherein the commander observes his environment (friendly and enemy forces, weather, terrain), processes the sensed information, compares his present state to the desired state, decides upon a course of action, and acts upon his/her decision.

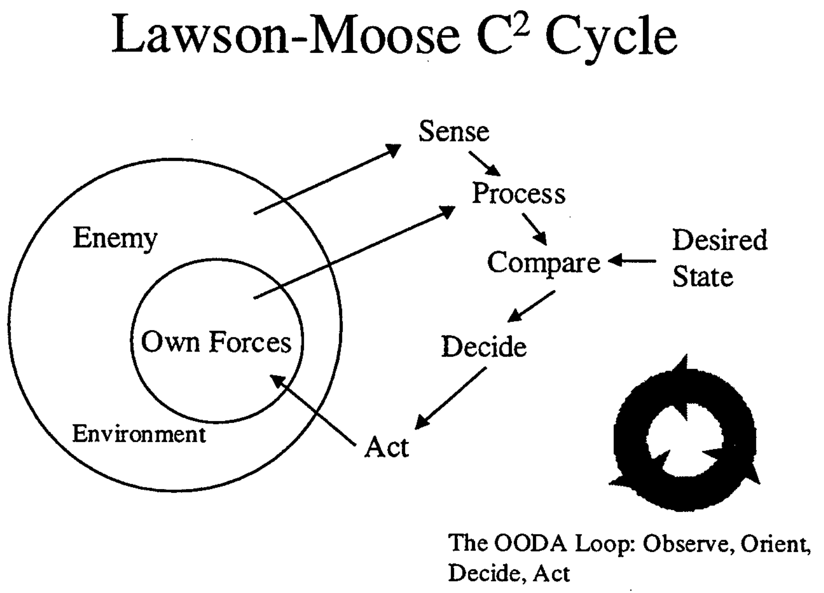


Figure 1. 1 Similarities between Lawson-Moose Cycle and the OODA Loop. After Ref. [PEND93]

This cycle demonstrates each of the three elements of command and control noted earlier: the process (planning, directing, coordinating and controlling forces in the accomplishment of the mission); the command function (the exercise of authority and direction by a properly designated commander); and the system (the arrangement of personnel, equipment, communications, facilities and procedures which are employed by the commander). Each of these elements, within the context of the decision cycle, offers an opportunity for a model to assist the decision maker in improving his or her ability to make the right decision, thereby lessening uncertainty in a timely manner.

In 1989 and 1990 the US Air Force Center for Studies and Analysis was using a raid simulation model, then-called C3ISIM, to study command, control, communications, and intelligence in various programmed scenarios [CASE91]. Developed for the US Army Space and Missile Defense Command (SMDC), C3ISIM (now known as EADSIM-Extended Air Defense Simulation) is an analytic, Monte Carlo[3], and deterministic model of joint and combined force air and missile warfare, used for scenarios ranging from few-on-few engagements to theater-wide applications. It is a workstation hosted, system-level simulation used to assess the effectiveness of air and missile defense systems against a spectrum of air and surface-launched threats. EADSIM models fixed and rotary wing aircraft, tactical ballistic missiles, cruise missiles, a variety of bombs, high energy weapons, infrared and radar sensors, satellites, command and control structures, sensors and communications jammers, communications networks and devices, and fire support in a dynamic environment.

As a result of the invasion of Kuwait, the Center for Studies and Analysis began adapting C3ISIM as an air planning support tool. A suite of powerful workstations was deployed to Riyadh along with a team of trained analysts. Populating the databases of C3ISIM was essential to successfully modeling the overall air campaign and its subsets. A key to the success of the effort, however, was that it was co-located with, and received extensive support from, joint military operators who were in the process of developing the actual Air Tasking Order (ATO). Despite the fact that these workstations were not connected to a wide area network (WAN) and were unable to take advantage of metacomputing or data mining, C3ISIM proved to be highly valuable in assessing potential sortie attrition, allowing planners to modify strike packages and compare results without risking lives and materiel.

Because this early version of the model was extremely resource intensive (three hours of real time air operations required eight hours of simulation run-time), C3ISIM lost some of its usefulness as a timely mission planning tool following the commencement of Desert Storm. The high pace of operations and the rapidly changing

[3] While EADSIM is described by the program's designer, Teledyne Brown Engineering, as a "Monte Carlo model", it is perhaps more accurate to describe it as a discrete-event-simulation model that employs Monte Carlo methods to generate random variates. A detailed discussion of Monte Carlo and discrete-event simulation is available elsewhere. [KELT91]

ATO caused analysts to turn their attention to "after-action" modeling. Post-mission analysis still involved the modeling of air missions, but was limited to specific target packages (vice the broad panorama of theater wide air operations).

EADSIM modeling methodology was improved as time passed both in-theater and via remote technical support, but the real potential of this decision support tool was perhaps never realized. How much more useful could this model have been with reduced processing time (made possible by metacomputing), flexible resource scheduling (through efficient distributed system management), and an adaptable model capable of reconfiguring to meet quality of service requirements in a changing environment?

> Advanced modeling and simulation (M&S) may integrate a mix of computer simulations, actual warfighting systems, and weapon system simulators. The entities may be distributed geographically and connected through a high-speed network. Warriors at all levels will use M&S to challenge their military skills at tactical, operational, or strategic levels of war ... [MSMP95]

In bringing the model to the Warfighter in a distributed, heterogeneous environment, resource management tools (discussed in Chapter II) that can address specific Quality of Service parameters are needed. These parameters might include a network subscriber's bandwidth allocation, application access priority (assigned by higher authority), preference for display of results (virtual reality, full motion video, high resolution images, graphical representations, text only, etc), preference for granularity and type of model chosen (stochastic, deterministic, time-stepped, event-driven), time constraints (perishability of material, timeline of pending military action), and subscriber's security access (assigned by proper authority and subject to authentication and verification). Such parameters can then be used to allocate the resources necessary to provide the requested Quality of Service or to offer the subscriber the opportunity to select an adaptable application capable of reconfiguring to meet the subscriber's needs in a changing environment.

Modeling and simulation have a role to play in each of the three elements of command and control: the command function, the C4I process and the C4I systems. One logical venue for such modeling would be the Global Command and Control System (GCCS).

> GCCS is the central C2 system for achieving information superiority in the Joint Vision 2010. It is an integrated, reliable and secure command and control system linking the National Command Authority (NCA) to the Commander in Chiefs (CinC) of the major commands down to the Joint Task Force (JTF) and Component Commanders. As the top level infrastructure for automated support to C4I operations worldwide, it is to provide a seamless battlespace awareness by exchanging data, imagery, intelligence, status of forces, and planning information... GCCS employs (sic) client/server architecture using commercial software and hardware and open systems standards. Currently, GCCS integrates SUN, HP, and PC products and operating systems with ORACLE and SyBase distributed relational database support. [ANTH98]

In order to take advantage of the distributed resources and metacomputing necessary to incorporate robust modeling applications into GCCS (or to make them available by commercial WEB browser over a secure network), it is essential that models be adaptable and capable of working in a distributed system. It is that adaptability that will make the use of such models practical as a tactical decision aid for the Warfighter in a distributed, heterogeneous computing environment.

B. SCOPE OF THIS THESIS

In order to support adaptive and adaptation aware applications (defined fully in Chapter II, Section C) the Management System for Heterogeneous Networks (MSHN) architecture includes a Client Library to intercept system calls and measure the resources required to run an application[4]. The Client Library is transparently linked with each application to allow for the gathering and processing of useful data from system calls. This process should be transparent to the user's application, should require minimal overhead, and should be accomplished without the need for application source code. As

[4] MSHN employs a method of intercepting an application's request for hardware before it reaches the operating system by wrapping the application within a composite library, referred to as the MSHN wrapper. The MSHN wrapper intercepts a system call, adds pre- and/or post-processing functionality for measuring resource usage, and returns the value of the system call to the requesting application.

shall be discussed in Chapter II, the data collected from these wrappers are used by the components of MSHN to establish resource requirements, resource status, and optimization criteria; to make scheduling decisions; and to monitor application performance in order to ensure adequate quality of service. A wrapper for this purpose was designed and demonstrated in a precursor thesis [SCHN98] and was proven capable of gathering an application's resource usage data without the need for that application's source code and without adding excessive overhead.

The objective of this research was to wrap a robust, C4I modeling application, representative of complex modeling applications currently in use by the Department of Defense, in order to determine the resources required to execute the application on a stand-alone workstation. Specifically, the goal was to determine resource usage required to run the application using both Monte Carlo simulation and deterministic simulation. The resources required would then be weighed against the confidence level of the outcomes obtained. MSHN needs this type of information to determine which version of an application to execute, in order to provide the best Quality of Service, while meeting operational deadlines.

C. MAJOR CONTRIBUTIONS OF THIS THESIS

The need for robust C4I modeling and simulation in support of the tactical commander, the "Warfighter," has been established in numerous Joint Warrior Interoperability Demonstrations, Fleet Battle Experiments, and Joint and service specific exercises (discussed in Chapter II B 2). However, to make the use of such models practical in a heterogeneous computing environment, the resource management concepts addressed by MSHN are needed. Key to the implementation of MSHN is the requirement for adaptive and adaptation aware models that offer different versions of an application to meet varying Quality of Service demands. Before scheduling algorithms can be tuned to support such applications, more needs to be known about the actual resource requirements of such models. This thesis provides the first such data gathered on a complex, contemporary, C4I/air defense model currently in use throughout the DoD, with conclusions drawn regarding the trade-offs of computing resources and confidence in simulation outcomes. The data from this thesis will be used in follow-on research intended to determine the distributions of these resources. Once the distributions have been determined, resource data collected from EADSIM simulations will be used to develop a MSHN application emulator (described in Chapter II).

D. ORGANIZATION

This thesis is organized as follows: Chapter II addresses the need for distributed systems in order to take advantage of metacomputing in a heterogeneous computing environment. It discusses three tools designed to support distributed systems, and the role of MSHN in integrating such tools to support adaptive and adaptation aware applications. Additionally, it discusses the most closely related work of which we are aware: Armstrong's collection and analysis of large grained resource usage by Numerical Aerodynamic Simulation (NAS) benchmarks [ARMS97, HENS99]. Chapter III describes EADSIM's technical and operational architecture. Chapter IV explains the concept of the MSHN wrapper, and the specifics of wrapping EADSIM run-time executables. Chapter V provides an explanation of why EADSIM was selected to represent an adaptive application. The EADSIM scenario used in the experiment is described in detail, as is the measure of effectiveness chosen to weigh simulation outcomes against resource requirements. Chapter VI provides a description of the experiment including its methodology, the computing environment in which it was conducted, the resource measurement data gathered from running EADSIM in both Monte Carlo and deterministic configurations and analysis of the resulting outcomes from those runs. Chapter VI also offers possible explanations of the overhead added by the MSHN wrapper and an evaluation of the measure of effectiveness described in Chapter V. The final chapter provides conclusions and suggestions for future research and its application to C4I.

II. THE ROLE OF MSHN AND RELATED WORK

We should expect to participate in a broad range of deterrent, conflict prevention, and peacetime activities. Further, our history, strategy, and recent experience suggest that we will usually work in concert with our friends and allies in almost all operations...Improvements in information and systems integration technologies will also significantly impact future military operations.... [JOVI95]

This chapter places MSHN into perspective by describing related middleware standards and projects and explaining why such middleware is needed in what has become known as "network-centric warfare." Additionally, it describes the previous research most closely related to this thesis, Armstrong's work [ARMS97] with NAS benchmarks.

A. THE NEED FOR DISTRIBUTED SYSTEMS

While the geographic setting, venue, scope, and magnitude of future military operations is uncertain, they will likely involve non-co-located Commanders-in-Chief and/or Component Commanders, each facing unique work-space challenges, forced to function in a heterogeneous computing environment with insufficient on-site computing resources and constrained bandwidth. Such is the nature of warfare, low intensity conflict, and operations other than war in the information age. It was, perhaps, this environment that VADM Cebrowski [CEBR98] had in mind when he coined the term "network-centric warfare." With limited on-site resources and geographically disparate command and control elements, full advantage must be taken of collaborative planning tools, common databases, and powerful off-site computing assets. This can only be accomplished through the use of distributed systems.

A distributed system is a set of computers, connected by at least one network, that do not share memory or a common clock. The goal of a distributed system is to cause a set of computers to appear to the user as a single, powerful virtual machine. This is true whether accomplished through the use of a distributed operating system (allocates fine-grained resources of the virtual machine to application processes), a resource management system (allocates single machines or groups of machines within the virtual machine to application processes, but allows each machine to run its native operating system), or a distributed computing environment (provides paradigm libraries or programming language support to facilitate the sharing of available resources). There are five primary advantages offered by distributed systems: resource sharing (hardware and software can be shared among computers); enhanced performance (higher throughput and speed through increased concurrency); improved reliability (through replication of data files and services, distributed systems can be made more fault tolerant); improved availability (some components may fail without affecting the overall performance of the system); and modular expandability (hardware and software can be added without adversely impacting existing resources). Due to the fact that network latency (unbounded message passing time) and heterogeneous architectures make a global clock and completely consistent shared memory infeasible, it is impractical for a distributed system to have a coherent, total view of the global state. Therefore, synchronization and consistency of state present challenges to any distributed system. This means that each of the advantages cited above has accompanying pitfalls that must be carefully considered when designing the distributed system and selecting global state algorithms to detect stable properties (i.e., process termination, deadlock, garbage collection). The need for consistent global states is no where more apparent than in dealing with discrete event simulation models that incorporate both event-driven and time-stepped events. EADSIM is such a model. [SING94]

B. THREE MIDDLEWARE STANDARDS AND PROJECTS FOR DISTRIBUTED SYSTEMS

> In the last few years, clusters of LAN[5]- or WAN- connected systems have become a reasonable and cost effective alternative to the use of expensive, dedicated monolithic high-performance systems. The notion of a metacomputer has been coined, denoting a 'network of heterogeneous, computational resources linked by software in such a way that they can be used as easily as a personal computer'. [BRUN97]

[5] Local Area Network

MSHN researchers do not foresee the results of their resource management system (RMS) research as a cumbersome piece of software requiring separate installation and maintenance. Rather, the outcome of MSHN's research may be packaged as a middleware-level standard. An RMS packaged in this way would eliminate the need for separate installation and could be consolidated with the services that distributed applications use most often. This section discusses middleware standards and projects of interest to MSHN investigators.

Middleware is software that sits between applications and lower level communication protocols and operating systems. The purpose of middleware is to provide a variety of services (i.e., naming, security, load balancing, resource brokering, and communication) to applications being executed in a distributed, and perhaps heterogeneous, computing environment. Through the use of middleware, applications written in different languages, executing via different operating systems, might access common services and achieve interoperation.

The Object Management Group (OMG) is a consortium of more than 800 companies, whose goal is to specify a middleware architecture consisting of an object request broker, common object services and vertical application domains. While it is not OMG's intention that this architecture specification control the way in which middleware technology is implemented, the consortium is interested in ensuring the interoperation of implementations currently being developed. Common Object Request Broker Architecture is the evolving middleware standard being nurtured by OMG. [DOLG99]

This section discusses CORBA and two other middleware tools, and how they relate to MSHN's research.

1. CORBA

In an organization as vast and complex as the US military, it is unreasonable to believe that computing environments in the future will be anything but heterogeneous. Technological progress, the cumbersome DoD acquisition system (different for each service), varying needs and working environments, a variety of networks and accompanying protocols, and the unfeasibility/unwillingness to simply discard proven legacy systems and architectures assure that any large internet/intranet must be built from heterogeneous hardware, software and operating systems. While heterogeneity in itself is not negative and may, in fact, be viewed as an asset to be leveraged, it does present challenges to software developers and contractors desiring to take advantage of heterogeneous networked systems. Heterogeneity creates the need for middleware that can enable applications to share objects, functions and types without causing extensive software re-work for developers, or complex work-arounds for users.

The Object Management Group (OMG) was formed in 1989 to develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environments. Since that time, the OMG has grown to be the largest software consortium in the world, and has developed the Object Management Architecture (OMA). The OMA consists of an Object Model and a Reference Model. The Object Model defines how objects can be described, and the Reference Model deals with interactions between those objects. In the Object Model clients issue requests for services to objects (much like a remote procedure call (RPC)). The implementations of these objects are hidden from the client. A key component of the Reference Model is the Object Request Broker (ORB), which facilitates communication between clients and objects (Figure 2. 1). Common Object Request Broker Architecture (CORBA) is the specification developed by the OMG that details the interfaces and characteristics of the ORB. In CORBA the terms "client" and "server" are roles that are filled on a case by case basis. A client for one request might be server for another [VINO96].

In CORBA, an application consists of one or more objects that may reside on the same or different platforms. An object provides service(s) that can be "requested" by a client (Figure 2. 1). Clients obtain services from an object by making "requests" (RPC-style requests, similar to a SEND operation, or by separate, deferred-synchronous, SEND/RECEIVE operations) that consist of an operation, the name of the object that will respond, zero or more parameters, and an optional request context. The object may or may not return results to a client, and will return an exception if an abnormal condition occurs. Code that is executed to perform a service is called a method. That is, a method defines the implementation details of an operation. Object Adapters are the run-time components of CORBA that sit between the ORB and the Object Implementations. Object Implementations may be written in a variety of languages and may exist in a variety of forms. With additional Object Adapters it is possible to support any style of object implementation. An Implementation Repository is used by the Object Adapter to provide run-time access to information about all currently available objects. [DUMA98]

Methods can be invoked statically or dynamically. In Static Invocation, a client's request is made via interface definition language (IDL) "stubs" on the client side, and the response is handled by IDL "skeletons" on the object side. The stubs and skeletons interface with the CORBA ORB. In Static Invocation, the IDL Client Stub converts data from the client's local data representation (type) to the Common Data Representation (CDR), which is platform and language independent. On the object's platform, the Object Skeleton executes the reverse operation. In Dynamic Invocation, requests are made via Dynamic Invocation Interface (which allows the IDL stubs to be replaced by separate SEND and RECEIVE operations). With Dynamic Invocation the developer is afforded more flexibility. In Dynamic Invocation, the Dynamic Skeleton Interface (DSI) may take the place of the Static Invocation Object Skeleton to accomplish data conversion at run time.

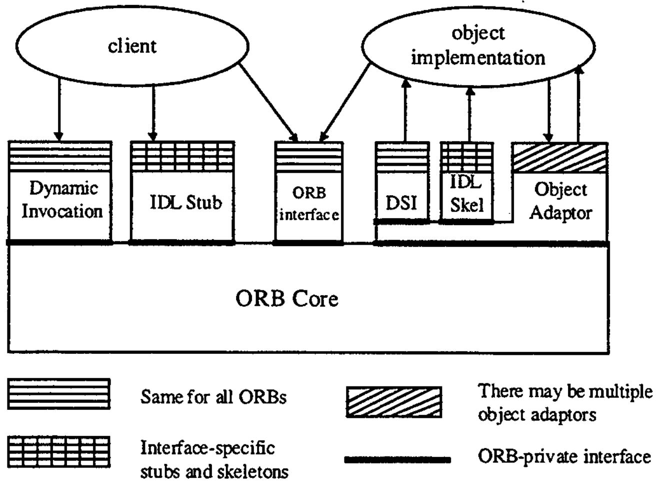


Figure 2. 1 Common Object Request Broker Architecture. From Ref.[Vino96]

CORBA supports two types of (method) invocation semantics: synchronous invocation and asynchronous invocation. Synchronous invocation is blocking. The client will invoke the method and block until it receives a response from the server (object implementation). With blocking primitives, the user buffer can be reused as soon as control is returned to the user program (when the message has been sent or an acknowledgement has been received). The RECEIVE primitive does not return control to the object executing it until a message has been copied into the buffer provided by that object. With synchronous primitives, a request (SEND) primitive is blocked until a corresponding RECEIVE primitive is executed at the receiving computer.

Asynchronous invocation is non-blocking. The client will invoke a method, continue its computation, and collect results as they arrive. With non-blocking primitives, the SEND primitive returns control to the requesting program as soon as the message is copied from the user buffer to the kernel buffer. The corresponding object that executes the RECEIVE primitive signals its intention to receive a message, provides a buffer into which the message will be copied and continues to execute. In CORBA the client can also make "one way" requests, continuing to execute while the object processes the request. [DUMA98], [SING94]

Through the IDL stubs, a client can use RPC-style semantics (synchronous), or by using Dynamic Invocation Interface (DII) a client can use SEND/RECEIVE semantics. Using DII allows a client to directly access the underlying request mechanisms provided by the ORB. Applications use DII to dynamically issue requests to objects without requiring IDL stubs to be linked in. The DII allows clients to make non-blocking "deferred synchronous" (separate SEND and RECEIVE operations) and one way (SEND only) calls. [SCHM99]

> The Global Command and Control System–Leading Edge Services (GCCS–LES) is a platform for the transfer of advanced applications from the Research and Development community to the Defense Information Services Agency's (DISA) GCCS service...The GCCS–LES is converging the architectures in the Modeling and Simulation (M&S) communities with Command, Control, Communications, Computer and Intelligence (C4I) architectures. [TBMC97]

DISA recognized the importance of CORBA middleware in realizing the potential of a heterogeneous, distributed system and incorporated it in the GCCS-LES. In fact, CORBA is now part of the Defense Information Infrastructure Common Operating

Environment (DII COE) standard web browser, and is finding increasing use in DoD C4I applications. The utility of CORBA lies in its ability to integrate diverse applications across a variety of networks and network protocols. CORBA's language independent IDL's allow interfaces to be used from a variety of programming languages, including COBOL, C, C++, Ada, Smalltalk, Perl and Java. CORBA-based applications are independent of network protocols so they may be run in a distributed system over a diverse network. These attributes ensure CORBA's usefulness in a heterogeneous command and control computing environment.

2. COMPASS

In 1994 the DoD Modeling and Simulation Office began sponsoring the US Navy's Common Operational Modeling, Planning and Simulation Strategy (COMPASS) Project based at SPAWAR Systems Center, San Diego.

> The goals of the COMPASS project were to: prototype the use of a common messaging environment to allow M&S services to better support C4I process; demonstrate the operational benefits to joint Warfighters of DCP tools to support M&S services for C4I/MP[6] systems; (sic) facilitate interoperability of M&S with C4I systems. [COMP99]

COMPASS sought a means for geographically separated Commanders, operational planners, and analysts to collaborate in a virtual environment as effortlessly as they would if they were physically co-located (Figure 2. 2). In order to do this, they would need distributed collaborative planning (DCP) tools that supported legacy systems. These systems include modeling and simulation, C4I and mission planning systems that were already in place. This collaboration would have to be accomplished in a distributed, heterogeneous environment accessed by joint services and coalition partners. These decision makers and mission planners would have access to a range of bandwidths, possibly limited local computing resources, definite time constraints, and most importantly the need to share a common image of the battlefield. The COMPASS

[6] Command, Control, Communications, Computers, and Intelligence/Mission Planning

project decided to pursue their goals through a combination of commercial off the shelf (COTS) and government off the shelf (GOTS) collaborative services integrated by middleware (Figure 2. 3).

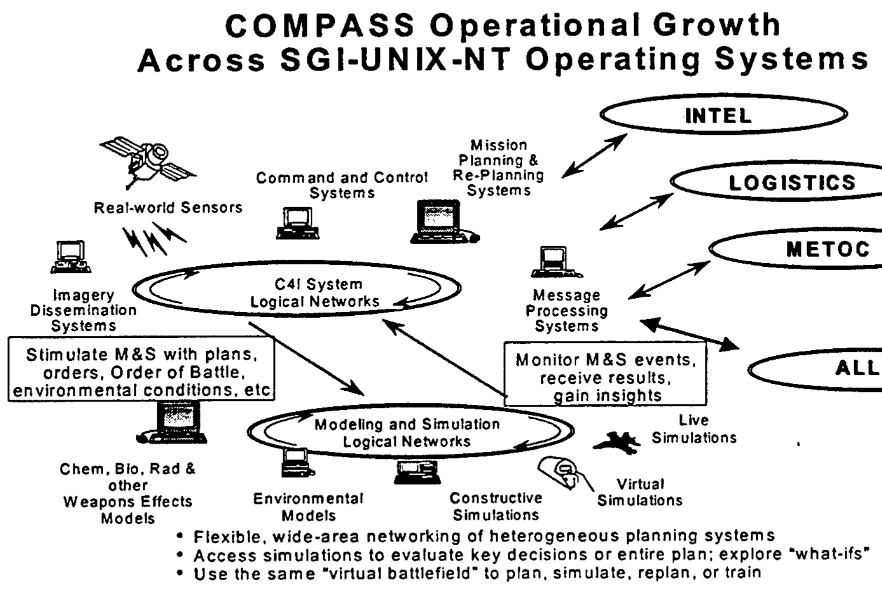

COMPASS Operational Growth
Across SGI-UNIX-NT Operating Systems
INTEL
LOGISTICS
METOC
ALLIED
Real-world Sensors
Command and Control Systems
Mission Planning & Re-Planning Systems
Imagery Dissemination Systems
C4I System Logical Networks
Message Processing Systems
Stimulate M&S with plans, orders, Order of Battle, environmental conditions, etc
Monitor M&S events, receive results, gain insights
Modeling and Simulation Logical Networks
Live Simulations
Chem, Bio, Rad & other Weapons Effects Models
Environmental Models
Constructive Simulations
Virtual Simulations
• Flexible, wide-area networking of heterogeneous planning systems
• Access simulations to evaluate key decisions or entire plan; explore "what-ifs"
• Use the same "virtual battlefield" to plan, simulate, replan, or train

Figure 2. 2 SPAWAR Presentation Slide of COMPASS. After Ref.[MCSW99]

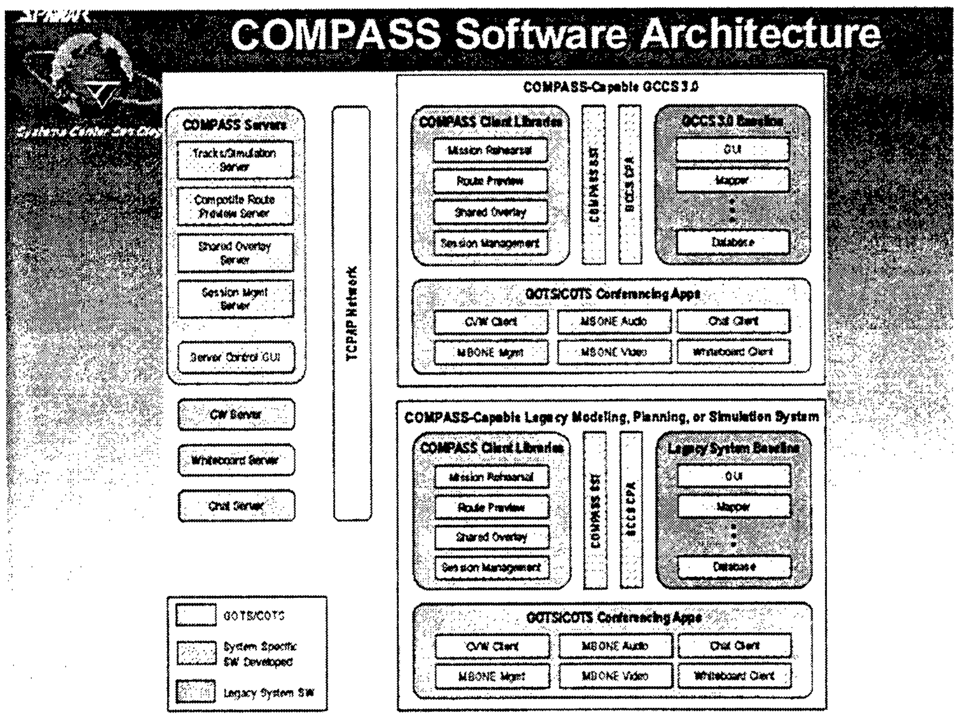


Figure 2. 3 COMPASS Software Architecture. From Ref. [MCSW98]

COMPASS middleware is a peer to peer, client-server architecture that offers core services to COMPASS-enabled systems accessing COMPASS servers through an Application Program Interface (API). COTS core services include: Whiteboard (enables sharing of graphics and images that can be cut and pasted, with tools for drawing and typing); Chat (allows low bandwidth exchange of typed information between multiple stations); Visual-Audio Teleconferencing/Video Interactive Conferencing (VAT/VIC) (VAT provides for video/audio teleconferencing via existing C4I/M&S applications, while VIC allows the sharing of video-based M&S products); and Collaborative Virtual Workspace (CVW) (establishes a "virtual DCP Conference center" in which participants can access collaborative planning sessions and COMPASS services through centralized and well organized virtual venues). GOTS core services consist of Session Management (provides the means to create, join or monitor collaborative sessions); Shared Overlay Management (enables the sharing of a variety of geo-registered overlays); Composite Mission Preview (allows planners to pre-view a complete, animated mission plan); Simulated Mission Rehearsal (permits the viewing of Distributed Interactive Simulation (DIS)-capable models and simulations); and Track Data Base Management Server (Track Server) (tracks from GCCS can be shared and updated with COMPASS-capable and non-GCCS COMPASS capable workstations). This suite of DCP services provides the capability of integrating a variety of modeling and simulation, mission planning, and C4I systems into a common view during the planning process by ensuring that users are working with displays that show common tracks and are updated from a shared data base manager with geo-registered overlays (routes, weapons effects, etc.). Through a session management facility, the COMPASS architecture provides start-up, recovery (re-establishment of system state following abnormal termination), and backup services. There are currently twelve M&S systems (including EADSIM) and eight C4I/MP systems that are considered COMPASS-enabled, having incorporated the middleware code necessary to access COMPASS services (Figure 2. 4).

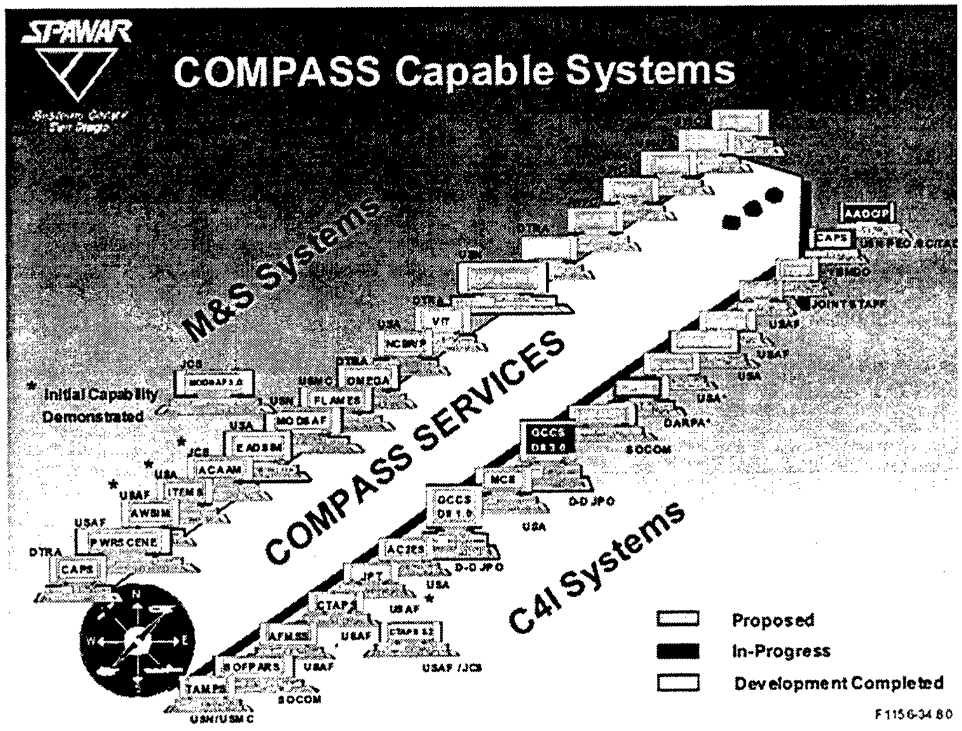


Figure 2.4 COMPASS Capable Modeling and Simulation and C4I Systems. After Ref. [MCSW98]

Since 1995 COMPASS has participated in more than twenty Joint and Service sponsored demonstrations and exercises, including four Joint Warrior Interoperability Demonstrations (JWID 1995, 1996, 1997 and 1998) and two Fleet Battle Experiments (FBE C, D). In each of these, COMPASS used metacomputing, reach-back and anchor desk concepts to support the Warfighter with robust distributed collaborative planning (in turn supported by modeling and simulation). Through these extensive operations COMPASS has repeatedly demonstrated not only the usefulness of COMPASS services, but the potential contribution of tactical modeling and simulation as an integral part of the C4I process. COMPASS is now transitioning from an R&D program to an operational implementation.

3. ENSEMBLE

The Ensemble system, developed at Cornell University, is a network architecture designed to support network and application adaptation to changes in environment, users, or application requirements. It is based on the use of protocol stacks to facilitate enhanced communications between applications (groups and group members) by supporting data replication, collaboration, or coordination. Ensemble is also capable of controlling or managing a network application in a manner transparent to the application developer. Elements of an application in a distributed environment include the users, the components of the application, the network, and the communication infrastructure and protocol. Adaptation can be achieved by reconfiguring any of these elements in response to a change in environment or requirements (Figure 2. 5). This reconfiguration might involve adding or deleting users from a group, changes in the communication infrastructure (bandwidth, protocol, security, etc.), or changes in the application (resources needed) itself. All of these responses to a changing environment require careful monitoring, control, and synchronization of the system state. Ensemble addresses this need through the use of a layered protocol architecture.

In Ensemble, micro-protocol modules are used to meet the communications needs of an application. Sliding windows protocols, fragmentation and re-assembly, flow control, encryption, group membership and message ordering may be stacked in a variety of ways to address these needs. Each configuration of an application and environment is served by a stack of these protocols. Assumptions are made each time a configuration is initiated. These assumptions are then monitored by "detectors" (which can be micro-protocols themselves) that sense changes in the environment (actually violations in the assumptions of the currently monitored configuration) and provide the protocol stack with enough information to form the basis of new assumptions in the event of a reconfiguration. The determination of when to reconfigure is based on crossing a preset threshold of violations for the given assumptions.

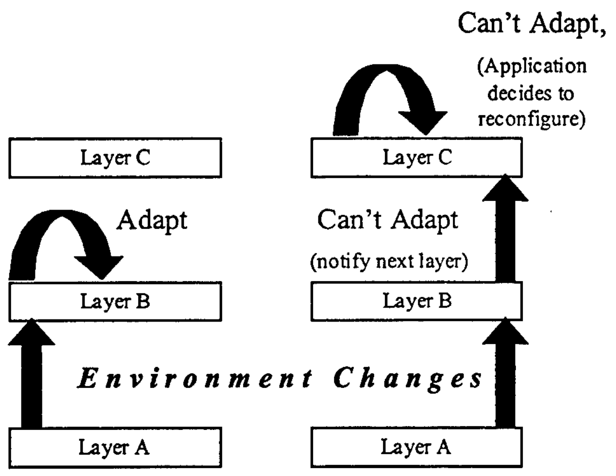


Figure 2. 5 When the Environment Changes, the Opportunity to Adapt is Passed Up the Protocol Stack Until a Layer Adapts or the Application is Notified (Requiring Reconfiguration). After Ref.[HAYD97]

In the Ensemble layered architecture, the lowest layer protocols attempt to adapt to environmental changes first. If they are unable to do so, they notify the layer directly above them. This continues until either one of the protocols on the stack is able to adapt, or the application itself is notified. Once notified of the changed environment, the application must decide whether, or how, to reconfigure (Figure 2. 5). A reconfiguration, or "adaptation," might involve the application adjusting its Quality of Service demands, dynamically modifying its interface with other applications, increasing or reducing its security requirements, increasing or decreasing its bandwidth requirements, or possibly even ceasing service to one or all application users. Each reconfiguration will result in a new protocol stack to support it (each "instantiation" of a Protocol Stack has a unique Protocol Stack Instance Identifier, PSI-ID).

Any reconfiguration in a distributed environment must address consistency of state which in turn is likely to involve finalization, synchronization and initialization (or start-up) of a new state. In Ensemble the mechanism that creates and installs a new stack across a set of application users when a reconfiguration occurs is called a Protocol System Protocol (PSP). The PSP, itself a stack of protocols within the application's stack, handles the finalization of the micro-protocols in the old stack, creates a new stack (assigning it a new PSI-ID), and starts the new protocol stack.

In order to reconfigure, the PSP selects the user with the lowest network address as a kind of "reconfiguration coordinator." The chosen coordinator (the user that has been selected) generates the new PSI-ID and broadcasts a finalize message to each user. This message contains a description of the new stack, the list of users, and the new PSI-ID. Each recipient builds the new stack of micro-protocols, registers the PSI-ID and passes a finalize event to the top layer of the old stack. When each layer of the old stack is ready to stop sending messages, the finalize event is passed down to the layer below. When the finalize event has been processed by the bottom most micro-protocol, a finalize--acknowledge message is returned to the coordinator. When the coordinator collects all finalize--acknowledge messages, it broadcasts a start message to the new stack. Each recipient in turn passes a start event to the bottom most layer of the new stack, discarding the old stack. The start event is passed up the new stack and discarded by the top layer. For fault tolerance, in the event a user is unreachable or there is a network failure, the coordinator will broadcast a new finalize message (containing the list of all users in the group). The coordinator awaits finalize messages from users of both the old and new stack. If the coordinator fails, the user with the next lowest network address broadcasts the finalize message. PSP will avoid deadlock; if necessary a user is capable of installing a stack with a single element and resuming operation. A MERGE micro-protocol may be used to locate concurrent PSI's for the same application, and to merge them. [HAYD97]

An increased dependence on resource sharing has created the need for more reliable means of managing and controlling applications in a dynamic, distributed environment. Ensemble addresses this need by providing layers of micro-protocols that pass events up and down a stack, responding only to messages of interest to their own layer (Figure 2. 6). A system of checks and balances is used wherein assumptions are determined when the stack is formed, and violation detection monitors watch for changes in the environment that could signal the need for reconfiguration (adaptation). Synchronization, finalization and consistent state reconfiguration are ensured (by implementing the Process System Protocol). The Ensemble system is one approach to making adaptive applications in a distributed system a reality. The potential exists for a system like Ensemble to incorporate middleware like CORBA or COMPASS and offer protocols that would enable adaptive modeling and simulation applications to span the most diverse, heterogeneous C4I network.

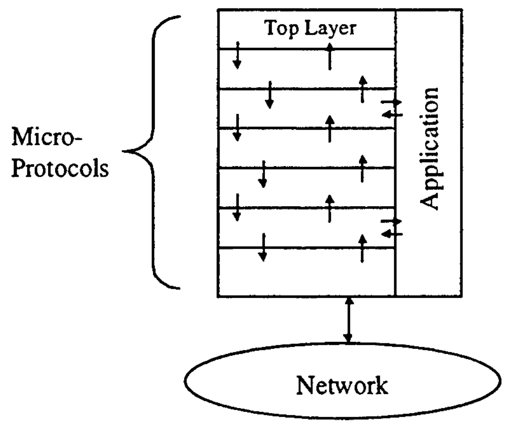


Figure 2. 6 The Ensemble Architecture. After Ref.[HAYD97]

C. THE ROLE OF MSHN

> Network intensive computing places unusual stress on conventional computer system management and operation practice... Because significant remote computing and storage resources may be necessary, standardized services for resource allocation and usage accounting are important. Other important issues are enforcing the proper use of network resources, determining the scale and quality of service available, and establishing priorities among the users and uses. Mechanisms are needed to address these issues automatically and dynamically. [NSTB96]

The Defense Advanced Research Projects Agency (DARPA)-sponsored Management System for Heterogeneous Networks (MSHN) project is a sub-component of the DARPA QUORUM program. The goal of this project, as well as the program at large, is to provide a resource management system (RMS) to enable both C4I applications and planning applications (indeed, any adaptive application) to use multiple sets of shared resources while accounting for dynamically changing priorities and environments. This RMS must be capable of providing each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences) in a heterogeneous computing environment in which many processes are competing for shared resources. Rather than establishing this RMS as stand-alone software, the MSHN architecture is designed as an integrated system, integrated with and incorporating a variety of distributed system tools (i.e., CORBA, ENSEMBLE, COMPASS, etc) to reap the maximum benefits from available resources.

In a military environment such an integrated system might mean offering the Commander the opportunity to select the most appropriate application, or version of an application, capable of executing within a specified time, at the proper security level, in order to deliver the best achievable answer within his stated time constraints. Specifically, applying this technology to robust C4I modeling and simulation applications would enable on-scene Commanders to simulate complex elements of the decision process in order to optimize the use of forces and materiel. The key to this implementation however, is the development of adaptive and adaptation-aware models and decision support applications that can be scaled to fit user-defined Quality of Service (QoS) parameters.

We recall that an RMS allows the user to view a set of computers as a single powerful virtual machine and does not provide micro-management of resources. Rather, as stated earlier, an RMS allocates single machines or groups of machines within the virtual machine to application processes, allowing each machine to run its native operating system. MSHN acknowledges that in a heterogeneous computing environment such as exists in the DoD, there will be little chance of an overarching distributed operating system controlling all resources such as network routing, protocols, and server memory allocation. It is MSHN's goal to simultaneously support multiple processes, each with a variety of QoS requirements while dynamically combining these requirements into a consolidated optimization criterion, then using this criterion to optimize the application of resources to these processes. In this way, MSHN will provide support for adaptive and adaptation-aware applications.

MSHN's use of the terms "adaptive" application and "adaptation-aware" application is best explained in terms of Quality of Service support. Adaptive applications exist in different versions capable of producing like results (though possibly offering varying degrees of Quality of Service). For instance, an adaptive application might have one version that executes on Linux, while others execute on Solaris, Unix or NT. MSHN would monitor the use of such adaptive applications and would be able to terminate one version and start another version from the beginning if it perceived the user's QoS requirements were not being met by the currently executing version. It is clear from this example then, that adaptive applications must be idempotent to allow an application to be started (in another version) without corrupting any resources (such as a database). An adaptation-aware application is similar to an adaptive application except that when terminated, it is not necessary to restart the new version from the beginning. Information about a previous state taken from the older version (that is about to be terminated) can be safely used in the newly started version. MSHN would thus monitor an adaptation-aware application as well as resources available elsewhere in the distributed system, and, upon finding resources to run a version of the application that could provide better QoS to the user, start the preferred version from the current state of the old version (then terminate the old version). Adaptation aware models could allow the user to "upgrade" to improved quality of service when resources become available without a costly loss of time or data. In a tactical environment, this means the transition to improved QoS would be transparent to the decision-maker, with no ill effect on mission planning or timing and potentially considerable positive effect.

1. MSHN Architecture

The primary elements of the MSHN architecture are the Client Library (CL), the Resource Requirements Database (RRD), the Resource Status Server (RSS), the Scheduling Advisor (SA), the MSHN Daemon (MD) and the MSHN Application Emulator (AE). The gateway of this architecture is the Client Library (Figure 2. 7).

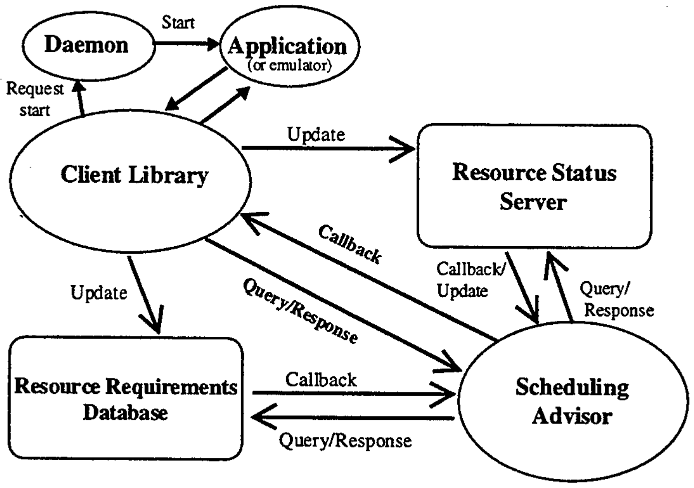


Figure 2. 7 MSHN's Conceptual Architecture. From Ref.[HENS99]

To avoid requiring a MSHN user to explicitly log into the MSHN RMS, it was necessary to intercept all calls to system libraries that would initiate new processes and to divert these calls to the MSHN CL. Upon receipt of a request to launch an application, the CL checks to see whether the application is on a list of applications that need not be managed by MSHN. If it is, then the request is passed to a local operating system for execution. If, on the other hand, it is an application that does require MSHN management, the CL invokes a scheduling request on the SA, sending along the QoS requirements defined by the user. The SA then consults the RRD, to determine what resources are required based on the user's QoS parameters. Afterwards, the SA polls the RSS to see what resources are currently available, and, based on this information and the optimization criteria, the SA decides where the process should execute and returns this choice of location to the CL. Based on input from the SA the CL then requests the Daemon located on the selected remote machine to execute the application (this will be accomplished through the use of CORBA, though not directly via the CORBA ORB). After the application has been launched, the CL is capable of responding to callbacks from the SA by requesting Daemons on other remote machines to launch preferred versions of adaptive or adaptation-aware applications. Daemons are also used to start the AE in order to simulate the running of applications without the accompanying overhead (to produce predictive statistics regarding resource requirements), or to monitor the status of resources not being used by MSHN applications. Key to the implementation of this architecture is the "wrapping" of user applications by the CL.

In order to provide MSHN services transparently to the user, the CL must intercept, pre-process and, when required, divert system library calls from user applications. This includes calls to `exec`, all socket calls, and calls to `open`, `close`, `read`, and `write`. By pre- and post-processing these intercepted calls to the operating system, the CL can determine the resources used by an application while it is running, store this information in the RRD, update the RSS as a process continues to execute, and call upon the SA to use current and historical data to determine where it is best to run the application in order to support its QoS requirements. It is the wrapping of applications that enables the CL to drive the functionality of all MSHN components. It is a goal of MSHN that this wrapping will eventually be accomplished by means completely transparent to the user. Although it is anticipated that this will be accomplished through the use of CORBA middleware, MSHN has already proven the ability [SCHN98] to wrap applications without requiring source code (object files however are still required). A more detailed explanation of MSHN can be found elsewhere [HENS99].

D. RELATED WORK

MSHN evolved from a scheduling framework called SmartNet. The goal of the SmartNet project was to improve performance by applying sophisticated scheduling capabilities to a set of compute-intensive jobs, each of which may require multiple processes, onto a set of heterogeneous computers. SmartNet predicted the expected run-time of a job on a particular machine based upon the wall-clock time required by previous executions of the job. The wall-clock time was partitioned according to compute characteristics [KIDD99]. A scheduling module leveraged the heterogeneity of a set of jobs and a set of computers to achieve enhanced performance. A detailed description of SmartNet is available elsewhere [FREU98, KIDD96].

Although SmartNet was designed to be used in actual production, the project did make significant research contributions. MSHN, on the other hand, is intended to be a research system that expands upon SmartNet's research by considering the overhead that job sharing resources have on mapping and scheduling algorithms, support of adaptive applications, and the delivery of good QoS to different sets of simultaneous users [HENS99]. While SmartNet focused on wall-clock time to determine estimated time to compute, this thesis discusses the monitoring of a variety of resources that impact QoS.

The work most closely related to this thesis of which we are aware is that of MAJ Bob Armstrong, a graduate of the Naval Postgraduate School, whose thesis investigated the effects that different run-time distributions have on the performance of SmartNet. Armstrong used NAS benchmarks to determine the types of run-time distributions that might represent selected jobs on selected machines. Once these run-time distributions were determined, their parameters could be reproduced by a SmartNet simulator that he built. Experiments with benchmarks were run with the executable program, input and output data residing on the executing machine, as well as with the executable and data being obtained from a file server. When the executable and data were obtained from a file server, experiments were run with both a heavily and lightly loaded server and network. The executable programs used in these experiments, which involved sorting algorithms, were run both using a single processor of a multiprocessor computer and using multiple processors on the same machine. [ARMS97] The focus of these experiments was on total wall-clock time. Finer grained data regarding cpu time, number of bytes transferred via interprocess communication, number of page faults, and amount of data held in local cache were not collected. As a follow-on to SmartNet and Armstrong's work, MSHN investigators have recognized the need to collect finer grained data, through the use of wrappers, in order to more closely estimate a variety of resource usage distributions.

E. SUMMARY

Section A of this chapter discussed the advantages of, and challenges inherent in, distributed systems. Section B described three middleware standards that are currently being investigated for their potential integration into a resource management system. Section C provided an overview of the MSHN project, and Section D discussed the related work of Bob Armstrong and the SmartNet project.

III. EADSIM

This thesis presents the methodology and results of wrapping EADSIM as a representative, contemporary military application. It is thought that more finely grained data collected from a real application, in this case from wrapping EADSIM, will be far more valuable than wall-clock data collected from benchmark programs. Resource usage data, collected from EADSIM using the MSHN wrapper, will be used to determine distributions for a variety of computing resources. This data can then be applied to follow-on research using the MSHN emulator to investigate the performance of the various scheduling algorithms.

Extended Air Defense Simulation (EADSIM), formerly known as the Command, Control, Communications, and Intelligence Simulation (C3ISIM) and also as the Theater Missile Defense (TMD)/C3ISIM, is a technology outgrowth of a computer-based modeling program developed in the late 1980's. EADSIM is an analytic, Monte Carlo modeling tool for joint and combined force air and missile warfare. It is useful for scenarios ranging from few-on-few engagements to theater applications. It embodies a workstation-hosted, system-level simulation used to assess the effectiveness of air and surface launched missile defense systems against a spectrum of air and surface-launched threats. A robust[7] C4I infrastructure for both aggressor and defender forces can be simulated, allowing planners to evaluate their own forces and enemy forces in either role. EADSIM is managed by the Testbed Product Office, Space and Missile Defense Battle Lab, US Army Space and Missile Defense Command as the executive agent for the Ballistic Missile Defense Organization (BMDO). EADSIM, selected as a primary simulation in the BMDO Tactical Missile Defense Cost and Operational Effectiveness study, is the primary mission level model for the Air Force, and was chosen as the primary single integrated operational plan (SIOP) analysis model for United States Strategic Command (USSTRATCOM). EADSIM is used by over 370 agencies, including all US services, DoD and other US government agencies, as well as by the

[7] Although in computer science "robust" is a software engineering term meaning that a program has been carefully engineered and tested to ensure that it is bug-free, this thesis uses the term in a more military sense. In this thesis, robust is used to mean a complex range of capabilities.

governments of the United Kingdom, the Netherlands, Israel, Australia, Republic of Korea, Japan, Spain, and NATO alliance countries. [SMDC97]

A. OPERATIONAL ARCHITECTURE

EADSIM displays scenario generation, preview, and post-execution information to the user through the use of a graphical user interface (GUI). Graphics provide full color terrain data with overlayed scenario icons, three dimensional displays of scenario playback from any location in the battlespace, previews of scenarios with flight paths, cross sections through terrain, attacker-to-target pairings, sensor intervisibility displays, overlayed text windows, and overlayed maps on terrain. The displays are easy to access and convey important, user-selected information.

The GUI provides a series of pull-down menus as well as many point and click windows to view specification and input data. Help screens are available as needed. These screens give a short description of the specific input area, including appropriate units and examples. All data can be added directly by highlighting a field using the mouse. The GUI offers graphics for generating, modifying, playing back, and analyzing scenarios. These graphics are capable of presenting simulation results in two dimensional and three dimensional displays simultaneously, from anywhere in the battlespace. They can also be used to produce tailored textual results of engagements, launches, kills, communications, and detections. Additionally, EADSIM offers a bounds checking feature that includes contextual and consistency checks.

EADSIM has the capability to interact (confederate) with other simulations using the Distributed Interactive Simulation (DIS) protocol standards, and the Aggregate Level Simulation Protocol (ALSP). It can be confederated with campaign level models such as the Corps Battle Simulation and Vector-In-Commander (VIC), with high fidelity models, and with virtual simulators.

B. SYSTEM ARCHITECTURE

EADSIM consists of three basic elements: simulation set-up; runtime models; and post-simulation analysis (Figure 3. 1).

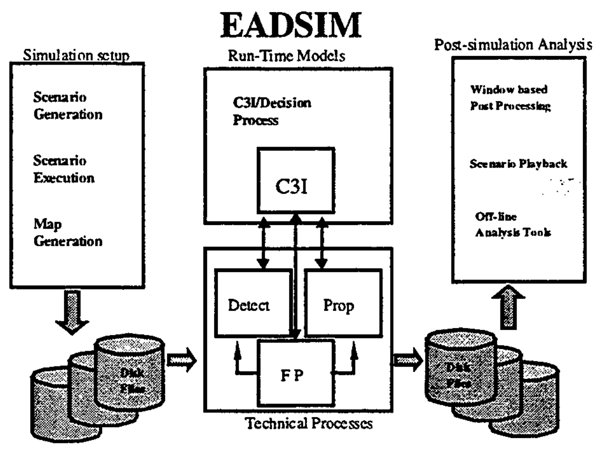


Figure 3. 1 The Three Elements of EADSIM. After Ref. [METH98]

A GUI executable provides the primary user interface for simulation set-up, post-processing and analysis tools. The four run-time processes are: Command, Control, Communications and Intelligence (C3I); Flight Processing (FP); Detection (Detect); and Propagation (Prop)(Figure 3. 2). The C3I process is event driven and serves as the simulation "driver," performing C2 decision processing, track processing, message processing, and engagement and weapons modeling for all platforms in the scenario. The other three run-time processes are time-stepped, receiving input events generated by the C3I process. Flight Processing maintains and updates the movement of aircraft, ballistic missiles, satellites, and surface platforms, providing "ground truth" to the other three processes.

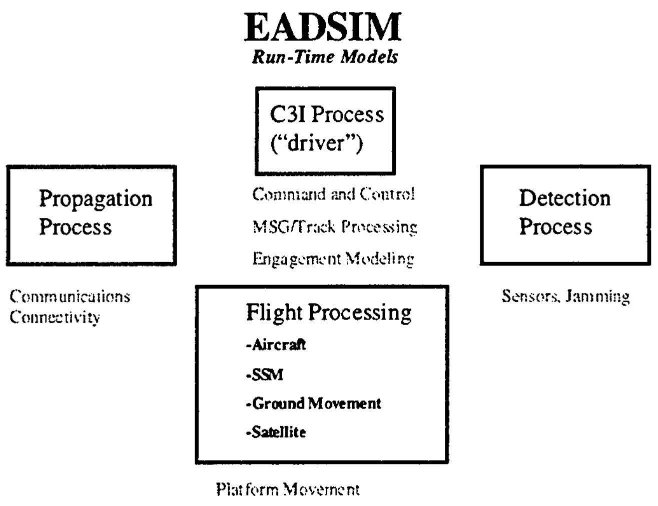


Figure 3. 2 The Four Run-Time Processes of EADSIM. After Ref. [METH98]

The Detection process models each sensor specified within the scenario provided and determines when detection of each participant by each sensor occurs. Propagation models communications connectivity and propagation. Propagation can be used to determine when message transfer occurs and the effects of information flow on the C4I decision cycle. Since the Propagation process is potentially very compute and resource intensive, scenarios may be run by the other three processes exclusively, ignoring the Propagation process. To minimize run time and computing resources, many EADSIM users choose not to include the Propagation process in their simulations. (Of note, the scenario we used in the experiments described in Chapter IV was provided in the EADSIM installation package and does not use the Propagation run-time process.)

Operations and platform types modeled include:

- Air Defense (SAM, artillery, Command/Control);
- Offensive Air Operations (CAS, SEAD, C2, TBM, Refueling, etc.);
- Attack Operations (Surveillance, C2, Intel gathering, movement, etc.);
- Multistage ballistic missiles, Air Breathers (aircraft, missiles, helicopters);
- Sensors (Radar, IR, SIGINT, IMINT, HUMINT, etc.);
- Jammers;
- Satellites;
- Early Warning;

- Generic Noncombatants;
- Communications (Networks, devices, messages);
- Electronic Warfare (full scope of capabilities);
- Terrain (masking, communications propagation);
- Weaponry (air to air, surface to air, air to surface, surface to surface); and
- Geographic Areas of Interest (MEZ, FEZ, AOR, etc).

C. TECHNICAL ARCHITECTURE

The execution of a scenario is performed by the four run-time models running in a multi-process configuration. Each process models some aspect of the scenario and exchanges data via interprocess communication during the scenario execution. Interprocess communication between the run-time processes is accomplished using sockets (Figure 3. 3).

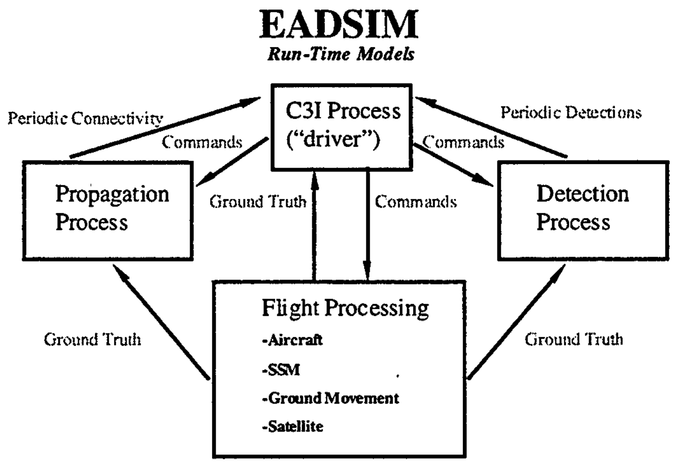


Figure 3. 3 EADSIM Interprocess Communication via Sockets. After Ref. [METH98]

Sockets are analogous to files and their use is analogous to file input and output (I/O) operations. When sockets are opened (using the system function call `socket( )`) they are named and associated with other sockets, thereby creating socket descriptors similar to file descriptors. Just as `read( )` and `write( )` operations are commonly used to modify and update files, `read( )` and `write( )` operations (perhaps more aptly described as "send and receive" operations) allow data to be transferred between sockets. Following a `close( )` operation, both file and socket descriptors are released back to the system. While files may be saved to memory and accessed later by a unique name, this is not the case with sockets. Socket names do not persist in memory following a `close( )` operation. After performing a `close(  )` therefore, an application must again open a socket(s) if it subsequently needs to transfer data. Simply put, the purpose of sockets is to allow two or more processes on a single computer, or on computers connected via a network, to communicate with each other. [QUIN96]

By using sockets for interprocess communication, EADSIM's run-time processes can run on multiple computers. EADSIM's sockets are blocking, that is they are established so a process will wait at the "read" operation until all data are received. This blocking mechanism allows for the proper sequencing of the processes. The timing and sequencing of the run-time processes are crucial to the ability of these processes to execute correctly. EADSIM primarily supports Monte Carlo simulation, but can also support deterministic analysis.

Although EADSIM is written in C, its coding is object-oriented-like. The object oriented nature of the code has made revisions and improvements more straightforward. Future versions of EADSIM are planned to integrate the functionality of the four run-time processes into one executable, simplifying the code and improving the application's performance on a single machine.

EADSIM executes on either Silicon Graphics Workstations with Silicon Graphics operating systems 5.3 through 6.3 or on Sun Workstations (SPARC Station 10, 20, or ultra series) with Solaris 2.3, 2.4, 2.5, or 2.6 operating systems. It requires 1GB Disk Space, 192 MB RAM, a CD ROM, a 4mm tape drive, and dbx (a debug utility used to trace the execution of a process).

1. Scenario Generation

Scenario generation in EADSIM is a complex iterative process in which laydown files are constructed from groupings of platforms (Figure 3. 4). Platforms are players in a scenario that reflect the attributes of prototype players. These prototype players, or system elements,

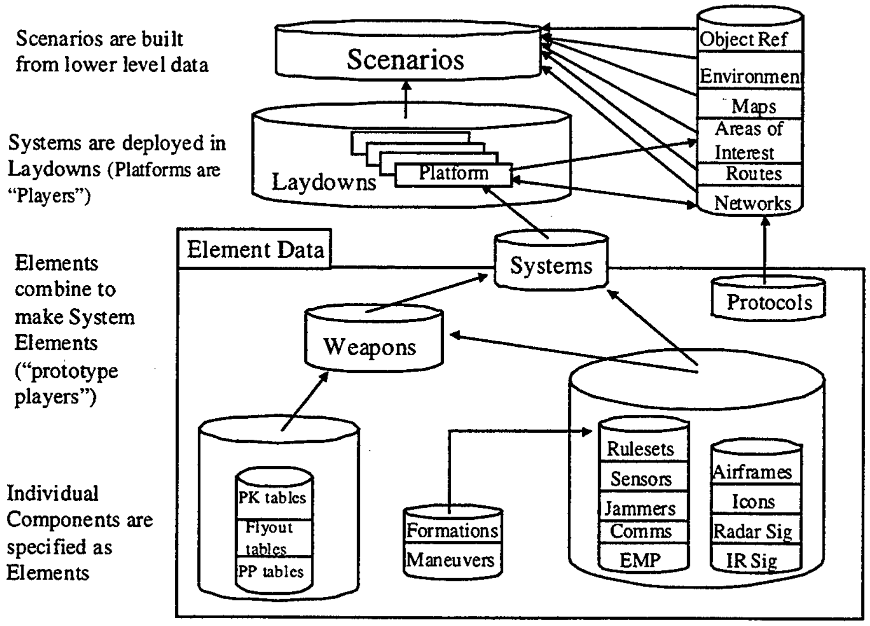


Figure 3. 4 Hierarchy of Data Builds an EADSIM Scenario. After Ref. [TELE98]

represent aggregates of elements (low-level component definitions). Element types include sensors (coverage and capabilities), airframes (flight dynamics), jammers (power, frequency, bandwidth, coverage), weapons (performance and effectiveness), fly-out tables (flight characteristics of interceptors), radar cross sections, infrared signatures, probability of kill (geometry and target dependent kill probability), power propagation tables (for directed energy weapon propagation), flight formations and relationships (command chain), icons for display, communication devices (power, frequency, bandwidth, coverage), protocols (message size, priority, and maximum life), rulesets (platform behavior), and classes (groupings of types of targets for identification). Complete details of the construction of scenarios are contained in the EADSIM v7.00 Methodology Manual. [METH98]

D. SUMMARY

EADSIM supports the "four pillars" of Theater Missile Defense (TMD) by modeling Active Defense, Passive Defense, Attack Operations, and Battle Management/C3I (including engagement logic, Command and Control structure, Communications networks, and protocols). By modeling Theater Ballistic Missile Defense and Air Defense in a dynamic environment, EADSIM offers analysts and training personnel enhanced insight into TMD architecture, battle management, system employment for maximum effectiveness, force structure analysis, and mission planning. [SMDC97]

The ability of EADSIM to serve as a timely decision support tool for the Warfighter has been demonstrated (Chapter II, Section B, Subsection 2 discusses EADSIM's inclusion in the COMPASS architecture) in Joint and Service sponsored exercises, Joint Warrior Interoperability Demonstrations, and Fleet Battle Experiments. Use of the model to support ATO preparation and planning during Operation Desert Shield/Desert Storm and its current use by more than 370 agencies worldwide attest to its potential as a decision support tool in a distributed, C4I planning environment. EADSIM is representative of modeling and simulation tools that can be made more accessible and timely through an ability to adapt to dynamic environments. EADSIM's ability to run deterministic as well as stochastic simulations made it a logical choice for measuring and comparing the computing resources needed to execute these two "versions" of the program. Additionally, EADSIM offers the ability to create elements with varying degrees of fidelity integrated with command and control chains consisting of varying degrees of complexity.

IV. EXPERIENCES WRAPPING A C4I MODEL

This chapter describes experiences with wrapping a C4I-modeling application, EADSIM. The reason for wrapping this application is two-fold: 1) to acquire resource usage information from a non-trivial, military, command and control application, and 2) as a proof of concept of the transparency of MSHN's wrappers. MSHN has developed a method of intercepting an application's requests for hardware services via the operating system in order to measure the computing resources required to execute the application [SCHN98]. This method is called "wrapping" the application and enables MSHN to measure computing resource usage without incurring significant overhead and without requiring access to the application's source code. MSHN's wrappers can be linked with application object code without requiring modifications to source, access to source, or modifications to the operating system. Therefore, it is unlike DeSiDeRaTa [UTAR96], Graze [MOOR95], and Pablo [REED96] which all require source modification. Unlike these tools, MSHN's wrappers obtain both the application's resource requirements and the current status of the resources by intercepting operating system calls.

Section A of this chapter provides a high level explanation of how MSHN wraps operating system calls. Section B discusses the methodology of wrapping the C4I-model, EADSIM, including steps taken to correct minor problems when encountered.

A. INTERCEPTING SYSTEM CALLS WITH "WRAPPERS"

Operating systems serve two primary purposes: to fairly allocate hardware resources that must be shared among applications and, to "beautify" the hardware by providing the user a higher level interface with which to access the computer's hardware resources. The interface between the user and the hardware takes the form of system calls. When an application needs a hardware resource (i.e., disk space or the network) a call is made to a user level library. This user level library, which is linked with the application, requests the hardware on the application's behalf by invoking a system call from the operating system. All UNIX applications, for instance link, with the C library, which then makes the system calls on their behalf.

Based on research done for the SmartNet project, MSHN investigators determined that to more intelligently manage resources in a distributed system, fine grained resource usage data would have to be automatically collected and analyzed by the resource management system (RMS). Specifically, MSHN's Client Library is responsible for collecting and distributing this data within MSHN. A method was needed that could gather data without incurring excessive overhead. That is, the method must not tax the very resources it was monitoring.

In his thesis [SCHN98], Matthew Schnaidt described in detail the method he used to intercept system calls, which was derived from a mechanism used by CONDOR [LIVN95], thereby providing a means to measure resource usage through classes within the Client Library [SCHN98]. This method involves "wrapping" an application within a composite library (that includes a modified C library), thereby allowing requests for hardware to be intercepted before they reach the operating system. Similarly, values that are returned by the operating system call can be analyzed by this modified library. The library of functions that perform this service, by linking with an application to be monitored, are referred to as an application "wrapper." An initial determination of which system calls to intercept in order to collect fine enough grained data for the resource management system to perform optimally was also addressed in Schnaidt's thesis. Experience with SmartNet demonstrated that the estimated time to compute (ETC), based on measuring wall-clock time, was insufficient to accurately predict an application's resource requirements, and hence its total run-time when executed in a different environment. To better understand the effect various resources have on the execution of a process, MSHN investigators identified key resources to monitor and the metrics associated with them (Table 1).

It was initially determined that the major resources to monitor include the total time an application controls the cpu (cpu time), the total main memory and cache memory used (measured in pages), local and remote disk access (measured in number of bytes), terminal I/O (measured in bytes and time spent blocked awaiting user input), interprocess communication and other network traffic (both measured in bytes transferred). In order to gather resource usage data on these resources, several system calls need to be intercepted by the Client Library wrapper. The system calls that need to be intercepted by MSHN's wrapper functions include `socket( )`, `connect( )`, `open ( )`, `close ( )`, `pipe( )`, `read ( )`, `write ( )`, and `exit ( )`. Additionally, in order to wrap an application's start-up a modified `MAIN( )` function was written. A detailed description of how the wrapper was written and implemented is beyond the scope of this thesis, but is provided elsewhere [SCHN98].

Resource	Metric
Total Run-time	Time
CPU	Time
Memory	Maximum memory used
Cache Memory	Maximum cache used
Local disk	Bytes read
	Number of reads
	Bytes written
	Number of writes
Network disk	Bytes read
	Number of reads
	Bytes written
	Number of writes
Network	Bytes read
	Number of reads
	Bytes written
	Number of writes
Local interprocess communication	Bytes read
	Number of reads
	Bytes written
	Number of writes
Keyboard input	Number of bytes
	Time blocked waiting for user input
Power Consumption	Watts

Table 1: Resources to Monitor. From Ref. [SCHN98]

B. WRAPPING EADSIM

The process of wrapping an application begins with identifying which system calls are to be wrapped. A copy of the C library functions corresponding to these system calls then needs to be modified (in the case of MSHN wrappers, the names of system call functions contained in the C library are simply converted to upper case to distinguish them from their wrapper function counterparts). Wrapper functions, by the same name as the original system call function, must then be written for each system call to be wrapped. These wrapper functions will invoke the original system call by using its modified (upper case) name, and will perform additional resource monitoring and measurement functionality as required. The wrapper functions are then linked with the modified C library resulting in a composite client library. The C run-time object file containing the function call `main( )` is modified in a similar manner (with the name `main( )` being changed to `MAIN()`). The MSHN Client Library and the modified C run-time object file are then linked with the application's object file(s). The wrapping of the application is then complete.

In order to wrap EADSIM as a proof of concept of MSHN's Client Library wrapper functions, it was necessary to obtain the application's object code[8]. Recall from Chapter II that MSHN's wrappers were designed to be transparent to the user, that is to be executed without requiring any modification to the user application's source code. With the US Army's Strategic Missile Defense Command providing release authority, Teledyne Brown Engineering (TBE) delivered the object and archive files (.o and .a files) for EADSIM v7.00, as well as a makefile. Using the README files provided with the MSHN Client Library source code, and the Tutorial on Wrapping System Calls originally written by Matthew Schnaidt [SCHN98] and later modified for use with the Solaris 2.5 operating system by MSHN staff member Shirley Kidd [SKID99], we modified the README-FIRST file (APPENDIX A) and proceeded to wrap EADSIM by following the steps noted in paragraph 3 therein.

We first created a modified C library, libMSHNC.a, and copied this into the syscall_lib directory that contained the files of the MSHN Client Library. Ensuring that the MSHN_types.h file defined variables for the Solaris 2.5 operating system (SUN5), we ran the libraryMakefile to compile the client library, which produced the

[8] Theoretically, it should be possible to wrap executables instead using the Executable Editing Library (EEL) [LARU95] tool developed by the University of Wisconsin's Paradigm project. However, doing so is the topic of another MSHN thesis.

MSHN_syscall_lib.o file. The MSHN_syscall_lib.o file was then copied into the EADSIM directory for later linking with the EADSIM object files. At this point we were unaware of modifications that would be required to the files MSHN_syscall_lib.cc, MSHN_Monitor_RRD_CLASS.cc, and libraryMakefile in order for the client library to successfully wrap EADSIM. To catch calls to `main( )`, we next modified the C run-time object file, crt1.o, and renamed it Mod_crt1.o. This file was also copied to the EADSIM directory.

After modifying the original EADSIM makefile provided by TBE to reflect local directory paths, we inserted code to link EADSIM's object files with MSHN_syscall_lib.o and included the "-###" command so that the C language compiler, "cc", would show each component as it was invoked without actually executing (operating in the verbose mode without producing executables for each process). The output from this modified EADSIM makefile was used to produce separate (c shell) csh scripts to compile and link each run-time process. These individual EADSIM-process csh scripts were then modified by substituting the C run-time object file (crt1.o) with the modfied C run-time object file (Mod_crt1.o). It was later realized that to successfully link the EADSIM object files (recall that EADSIM is written in C) with the MSHN_syscall_lib.o file (written in C++), the makefile used to produce MSHN's syscall_lib.o object file would have to explicitly add a reference to the directory containing the stdc++ library to the list of directories to be searched by the linker.

EADSIM v7.00 was installed, from the production CD, on a SUN SPARC 20, a SUN Ultra One, and an SGI Indy workstation. The modified EADSIM-process csh scripts produced executables that appeared to be successfully wrapped. The four run-time processes provided with the installation CD (C3I, FP, Detect, and Prop) were then replaced with the wrapped run-time executables. These executables were used to run the Demo300 scenario provided in the EADSIM v7.00 installation package (use of the Demo300 scenario is described in more detail in Chapter V). While the simulation executed successfully, the wrapper did not initially produce output files as expected. Troubleshooting and debugging were needed.

After extensive debugging of the MSHN client library source code and EADSIM csh scripts, it was learned that the following modifications were needed: (i) we needed to explicitly add the "libstdc++.a," the "libstdg++.a," the "libm.a," and the "libc.a" to the library Makefile (APPENDIX B) used to produce MSHN's syscall_lib.o object file so that C++ language symbols used in the Client Library would be defined in the EADSIM executables at compile time; (ii) DEBUG statements defined in MSHN_Monitor_RRD_Class.cc, that had been added to the `write( )`, `_write( )`, and `__write( )` functions after the program had undergone testing, were causing segmentation errors and needed to be removed; and (iii) the correct invocation of the Unix `–ps` (process status) command had to be incorporated into the `getPsData ( )` function of the MSHN_Monitor_RRD_Class.cc code (APPENDIX D). Once these changes were made, csh run-scripts were developed to properly reset the random number generator seed for each run of the Demo300 scenario (APPENDIX E), and another run-script was written to transfer the data from each run (both simulation outcomes and wrapper output) to a separate directory for later analysis (APPENDIX F and APPENDIX G).

During the debugging process, it was immediately evident that functions deriving from the C++ `iostream` library were not being recognized (nor were any of the functions from the C++ `strings class`). The `ostream` class object `cout`, for instance, did not output debug statements to the screen. The C language function, `printf ( )`, however, did perform as expected. It was determined that the original files in the MSHN Client Library, written in C and C++, had always been compiled (and linked) with C++ language test programs, using a C++ compiler which automatically linked the MSHN_syscall_lib.o object file with the libstdc++.a library. However, when attempting to link the MSHN_syscall_lib.o object file with the EADSIM object files (compiled using a C compiler), to produce an executable, the C++ libraries were not being searched, and symbols being referenced in the MSHN_syscall_lib.o file remained undefined. This was verified by running the print name list of an object file (`nm –f`) utility on the EADSIM wrapped executables. The `nm` utility displays the symbol table of each ELF object file that is specified by the input file argument and will report if no symbolic information is available for a valid input [BERK91]. By running the `nm –f` command on the EADSIM executables it was discovered that several C++ symbols were undefined. Using the g++ compiler with the –v option (verbose mode) we compiled a toy program and noted that the libstdc++.a library, the libg++.a, the libm.a, and the libc.a libraries were referenced in the output. We then found the local path to these libraries and explicitly referenced them, using the –L command, in libraryMakefile2 (APPENDIX B). By recompiling the MSHN Client Library with this makefile, we produced a MSHN_syscall_lib.o file that, when linked with the EADSIM object files, produced an executable with statically defined C++ symbols. The wrapper then successfully printed debugging statements to the screen and produced output files as expected.

Apparently, after the MSHN Client Library had been successfully tested by Matthew Schnaidt[SCHN98], DEBUG statements were added to many of the functions in the MSHN_Monitor_RRD_Class.cc file to assist anyone planning to implement the Client Library in the future. It was found, however, that when DEBUG statements were defined within the MSHN_Monitor_RRD_Class.cc, running the wrapped executables caused a segmentation fault. This segmentation error was due to recursive calls to `write()` when invoking `cout` within the `write()`, `_write()` , and `__write()` functions in the MSHN_syscall_lib.cc file. The following lines were removed from these functions:

```
#ifdef MSHN_DEBUG
        cout << "CAUGHT A __write()" << endl;
#endif
```

This solved the segmentation error problem and provided a lesson on attempting to invoke a `write()` system call from within a `write()` function.

Once the changes were made to the Client Library makefile and to the MSHN_syscall_lib.cc file, the wrapped executables produced output files containing resource usage data. However, this data clearly was incorrect (identical output values were being produced by two of the three functions). There was something in the `getPsData ()` function, in the MSHN_Monitor_RRD_Class.cc file, that was not returning correct values. The problem was found to be in the invocation of the report process status `(ps)` command. The `ps` command prints information about active processes [BERK91]. The original MSHN_Monitor_RRD_Class.cc code invoked the `ps` function using the group list (`–g`) option, which prints information about all processes in a group list. The ps utility is used in the getPsData () function to determine virtual memory in kilobytes used by each wrapped process (-vsz), the number of physical pages in memory (-osz) for each wrapped process, and the number of pages of memory present in the resident set (-rss) of each wrapped process. It was determined however, that the –g option did not return data for each of the three executing processes. The –g option was replaced with the process list (–p) option, which returns data only on specified process ID numbers (pid) listed in the process list. With this modification made to the getPsData() function in MSHN_Monitor_RRD_Class.cc, the MSHN Client Library was re-compiled and the MSHN_syscall_lib.o file was relinked with the EADSIM object files. This produced the desired system usage data for each wrapped process.

One additional change to the code was made before recompiling the MSHN Client Library. The MSHN_syscall_lib.cc file originally written by Matthew Schnaidt [SCHN98] was intended to monitor processes communicating via network interprocess communication (IPC) as well as local interprocess communication. Because EADSIM's run-time processes would be run on one workstation, it would not be necessary to measure network latency and throughput by monitoring network IPC. To avoid adding overhead needlessly (by checking each connect( ) and accept( ) system call to see whether it was a local or network IPC), since all connect( ) and accept( ) system calls made by the EADSIM run-time executables would be made via local IPC, code was modified within the acceptWrapper( ) and connectWrapper( ) functions to ensure that all interprocess communication would be interpreted as local IPC (APPENDIX C). An alternate approach to reducing overhead is discussed in Matthew Schnaidt's thesis [SCHN98], in which he suggests that follow-on research might incorporate Synthetix tools, to dynamically optimize the MSHN Client Libraries at run-time. Synthetix tools are described in detail elsewhere [PUCA96A] [PUCA96B].

C. SUMMARY

Section A of this chapter discussed "wrapping," a method of intercepting system calls in order to monitor resource usage. Section B described the steps taken to wrap

EADSIM's four run-time processes, C3I, FP, Detect, and Prop. Problems identified through debugging were discussed, and solutions to these problems were explained.

V. PROVIDING GOOD QUALITY OF SERVICE WITH LIMITED RESOURCES IN A C4I MODELING APPLICATION

Section A of this chapter discusses the reasons EADSIM is a logical candidate to consider when determining whether good Quality of Service (QoS) can be provided when resources are limited. Section B describes the air and missile warfare scenario used in our experiments. Section C discusses the measures of effectiveness (MOE) chosen to compare the outcomes from the deterministic simulation with the outcomes from the Monte Carlo (stochastic) simulations.

A. WHY EADSIM?

Recall from Chapter II that MSHN's goal is to provide a resource management system (RMS) to enable applications that contend for shared resources to obtain good QoS while accounting for dynamically changing priorities and environments. Operating in a heterogeneous computing environment in which many processes are competing for shared resources, an RMS such as MSHN, must be capable of providing each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences). The MSHN architecture is designed to be integrated with, and to incorporate, a variety of distributed system tools (i.e., CORBA, ENSEMBLE, COMPASS, etc) in order to take advantage of all available resources. Applying this technology to C4I modeling and simulation applications would enable on-scene Commanders to simulate complex elements of the decision process in order to optimize the use of forces and materiel. In order for the RMS to offer this flexibility in response to user defined Quality of Service parameters, modeling and decision support applications must be adaptive. That is, they must exist in different versions capable of producing like results (though possibly offering varying degrees of Quality of Service).

EADSIM is an air and missile warfare modeling and simulation application that offers great flexibility in (i) the areas modeled, (ii) the capabilities of the platforms simulated, and (iii) the method of simulation (deterministic or stochastic). When building a scenario, each platform (i.e., aircraft, missiles, sensors) is modeled individually as is the interaction among platforms. A robust range of characteristics is offered for each platform, allowing the scenario developer(s) the opportunity to select the granularity they need to model in order to produce a result with sufficient fidelity to suit their purpose. Users may chose to simulate the Command and Control (C2) decision process and the communications among platforms on a message-by-message basis, simulating message traffic flow through command networks and the impact this flow has on the decision process. Intelligence gathering is also modeled, as is the flow of intelligence disseminated to the commanders making operational decisions. All of these capabilites can be applied to both offensive and defensive operations, allowing analysts and planners to evaluate strike and counterstrike scenarios textually, in 2-D playback and/or 3-D playback. It was EADSIM's flexibility, and potential as an adaptive C4I application, that made it a logical choice as a proof of concept for MSHN.

B. THE DEMO300 SCENARIO

Due to the complexity inherent in designing and building a realistic scenario that would exercise many of EADSIM's capabilities, it was decided to use one of the demonstration scenarios provided with the EADSIM v7.00 installation package for our experiments. Discussions with Teledyne Brown Engineering representatives led to the selection of the Demo300 scenario, due to its broad range of platforms modeled, its plausible force layout and mission objectives, and the short duration (wall clock time) of each simulation. The only drawback noted in choosing Demo300, was that the scenario did not model communications propagation (did not use the "Prop" run-time process). It was explained to us by the TBE engineers that EADSIM users frequently elect not to execute the propagation process because of its considerable computing resource requirements. Communications in Demo300 were then considered "perfect" with network latency, contention, and congestion not being simulated.

Demo300 simulates a large offensive air strike with a leveraging strike of tactical missiles. Air launched cruise missiles and lofted trajectory TBMs are targeted against Command and Control (C2) nodes--for both SAMs and Defensive Counter Air (DCA), and the Intelligence Center, while depressed trajectory TBMs are targeted against the Air

Base itself (Figure 5. 1). With SAM sites at least partially suppressed, offensive air-to-air fighters can engage defensive counter air fighters and divert them from the offensive ground attack aircraft that are to follow (Figure 5. 2). This leveraging strike is intended to soften the defenders' ability to counter the bulk of the attack conducted by ground attack aircraft. A stand-off jammer aircraft attempts to blind the defenders' sensors and disrupt communications links.

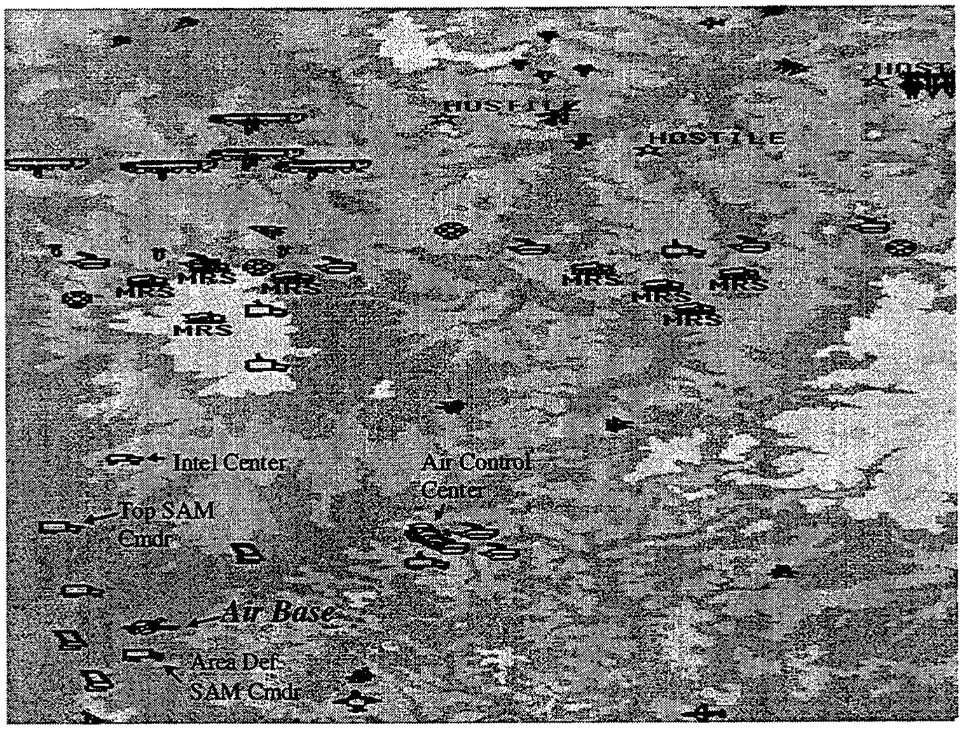


Figure 5. 1 Offensive Missile Targets. After [EADS98]

To counter this attack, a number of actions will be taken, utilizing a panoply of defensive and counter-offensive assets. Early warning and ground surveillance aircraft, battlefield C2 sensors, a low-observable, reconnaissance fighter aircraft, and an intelligence collection satellite will provide alerting information and target cueing for counter-offensive operations (Figure 5. 3). SAM and DCA C2 nodes will prioritize and check targets for engagement based on the available consolidated air picture, areas of responsibility, capabilities of assigned assets, and perceived threat to the assets being defended. SAM and DCA C2 nodes will perform deconfliction in the case of dual or multiple engagements of a single ingressing missile or aircraft (Figure 5. 4). While the air defense operations are ongoing, ground targets will be selected for counter-offensive operations.

Aided by intelligence, reconnaissance and surveillance assets, the defending force will target the attacking force's ground components that are supporting the attack. These ground targets would include ballistic missile launchers, sites occupied by these launchers, and the command chain controlling the launchers (Figure 5. 5). Hostile Division headquarters would also be targeted. Defensive force assets used in this counter offensive consist primarily of mobile rocket systems (MRS).

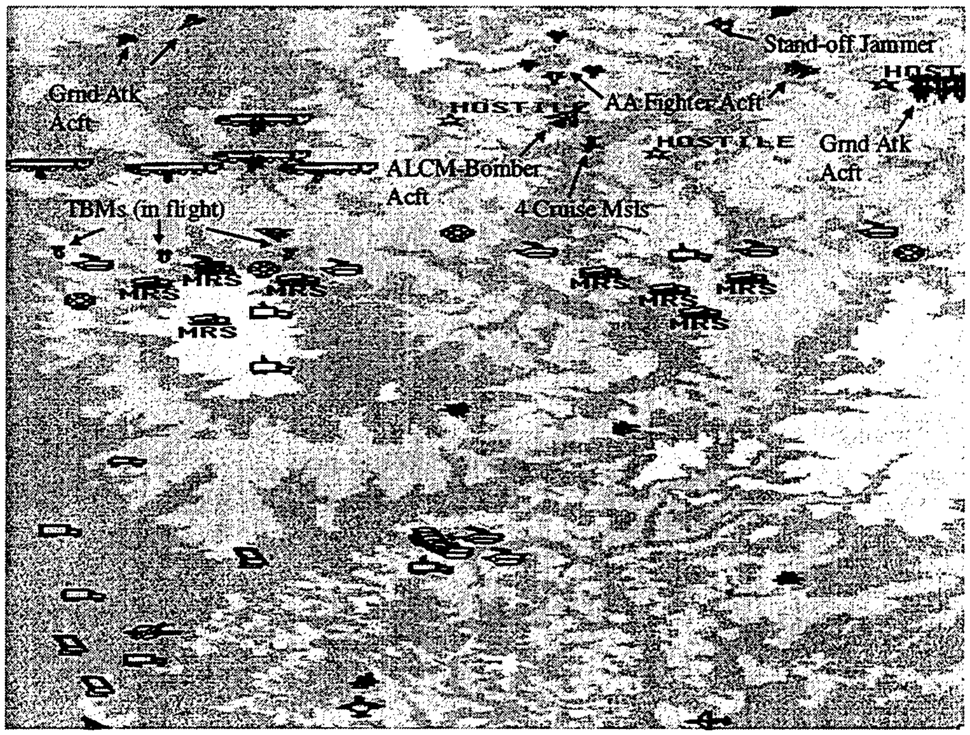


Figure 5. 2 Offensive Strike Aircraft and Tactical Missiles. After [EADS98]

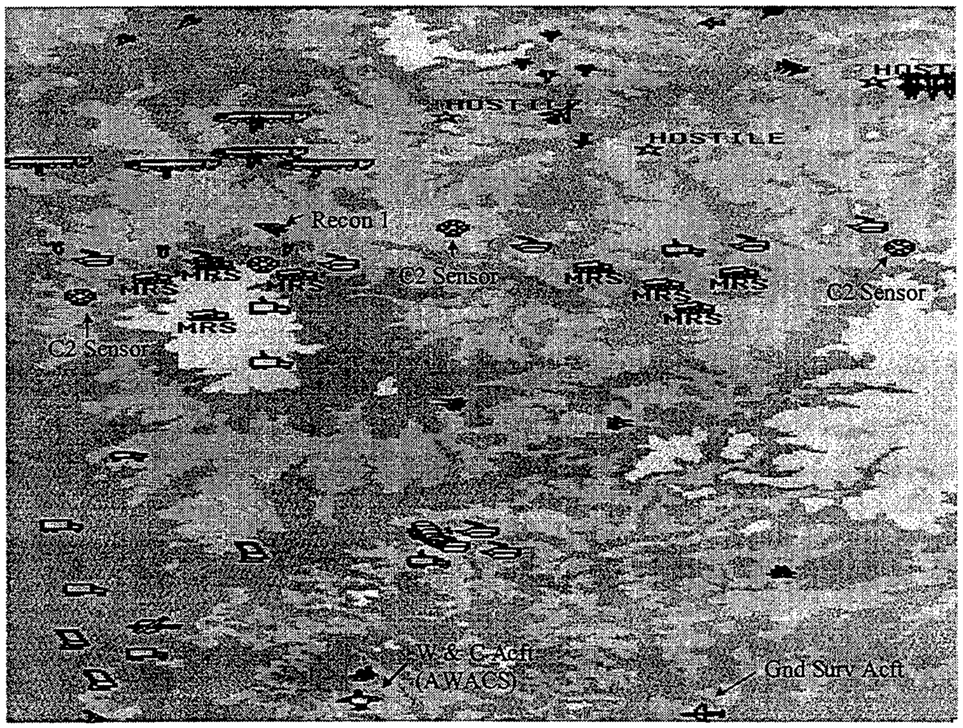


Figure 5. 3 Defensive C2 Sensors, Reconnaissance, Surveillance and Early Warning Aircraft. After [EADS98]

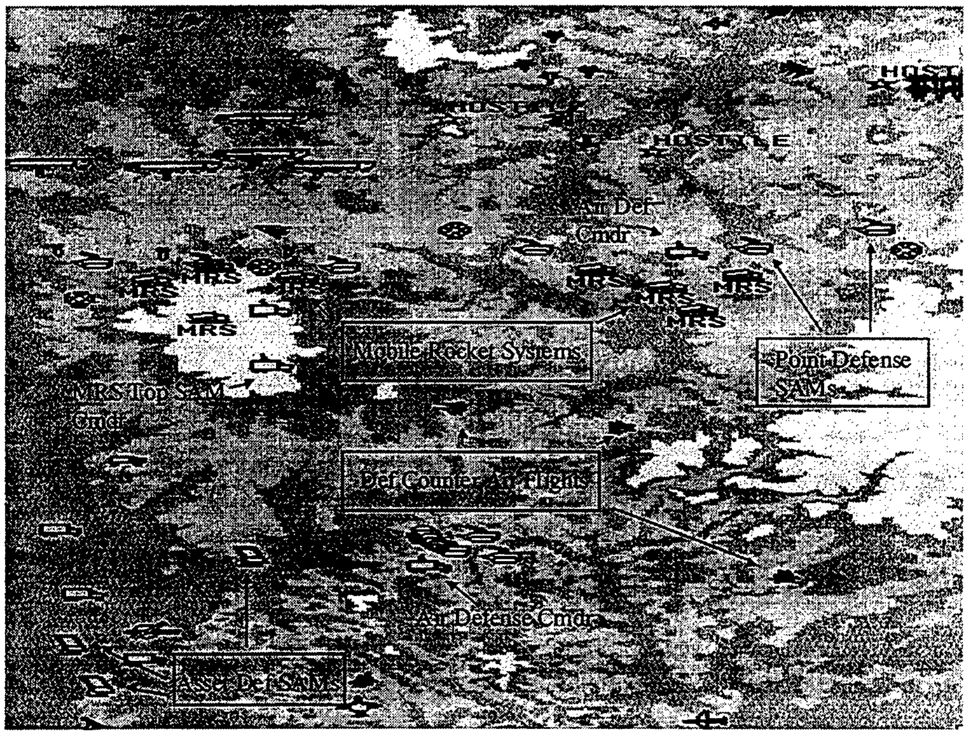


Figure 5. 4 Deconfliction is Coordinated Among the Defensive SAM and DCA Commanders. After [EADS98]

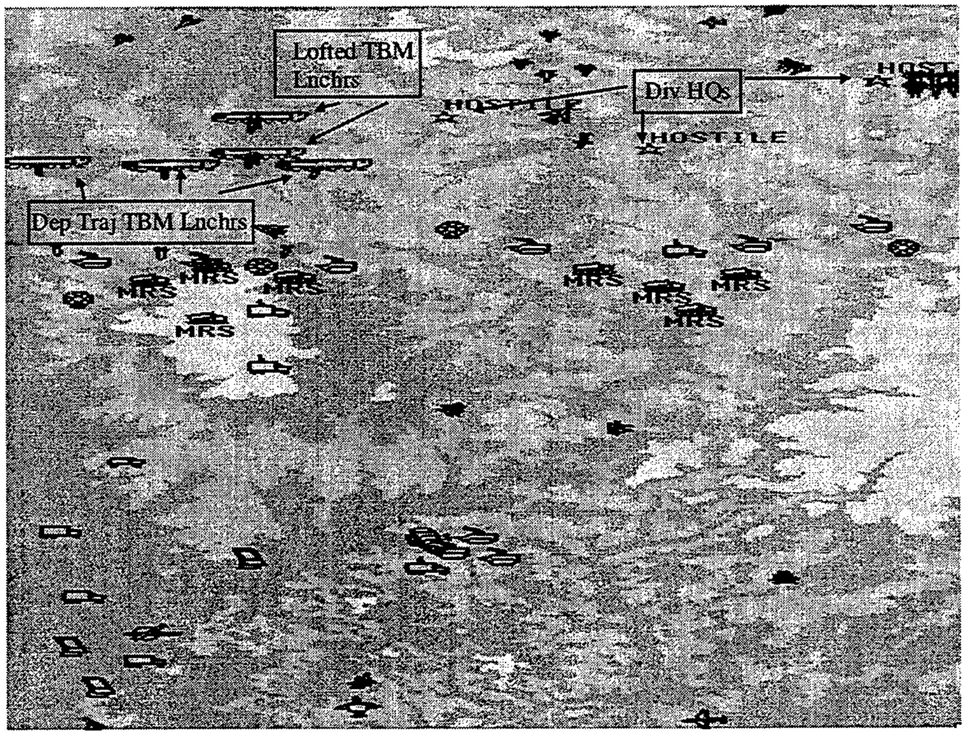


Figure 5. 5 Counter Offensive Targets. After [EADS98]

C. CHOOSING A MEASURE OF EFFECTIVENESS

Section C of Chapter II and Section A of this chapter, have discussed the fact that fundamental to MSHN's design as a resource management system is the capability of providing each subscriber process with its desired Quality of Service (QoS). QoS considerations commonly sited include security, deadlines, user priorities, and preferences. To military commanders and planners, these QoS considerations represent very real constraints that vary with the situation and may mean the difference between success and failure, life and death.

MSHN investigators are currently developing a framework for a performance measure that can be used for scheduling resources in a distributed heterogeneous environment. Such a performance measure would need to assess the ability of the RMS to simultaneously satisfy, to varying degrees, QoS requirements. QoS attributes considered critical to analyzing and evaluating a system's performance are priorities, versions (which represent a user's preferences), deadlines, and security. These attributes are defined, and their role in developing an optimization criterion called the Flexible Integrated System Capability (FISC) ratio is discussed in detail elsewhere [JONG99].

In a military environment, timeliness, security considerations, and priorities dictated by the operational situation greatly influence the decision-making process. While a commander or planner may have little control over these variables, it is rarely the case that decisions are *faits accomplis*, offering no opportunity to apply heuristics to alternative courses of action. Just as EADSIM is one tool intended to be used in the evaluation of alternative courses of action, it is an example of an application that could potentially be used in several variations, providing the user the opportunity to state a "preference" for one version over another. As stated in Section A of Chapter II, the key to implementing an RMS that can support QoS requirements in a distributed, heterogeneous computing environment, is the availability of adaptive, or adaptation-aware, applications. It is not the purpose of this thesis to examine the potential of EADSIM as an adaptive application. Several characteristics of EADSIM allow the application to provide the user with a variety of choices, in order to support the constraints of the decision making environment.

Among the characteristics of EADSIM that may be manipulated to support varying time and resource constraints are (i) the application's ability to display the scenario playback in either 3-D or 2-D graphics or to simply provide simulation results through a variety of tailorable, textual reports; (ii) the ability to input varying degrees of fidelity for each platform's characteristics, thereby "scaling" the model's scope; (iii) the ability to model communications propagation with varying degrees of fidelity, or to simply model "perfect communications," avoiding the considerable computing resources needed to execute the propagation run-time process; and finally, (iv) the ability to execute the application as either a stochastic or a deterministic simulation.

A primary objective of this thesis was to wrap a complex C4I modeling application in order to determine the resources required to execute the application on a stand-alone workstation in two different configurations, or "versions," and to compare the results obtained from each model configuration against the computing resources required to run them. We therefore sought a reasonable measure of effectiveness with which to compare different "versions" of EADISM.

In order to make our proof of concept as manageable as possible we chose to wrap the four run-time processes (despite the fact our experiment did require the use of one of these). As discussed in Section B of this chapter, we chose to use a demonstration scenario provided with the EADSIM V7.00 installation CD. The use of the Demo300 scenario, which modeled perfect communications propagation (and therefore did not use the prop run-time executable), eliminated the possibility of comparing a version of the simulation running the propagation executable with a version that did not. However, even this "example" was quite complex, corresponding to more than 600 pages of scenario platform laydowns.

One aspect of EADSIM however, made it particularly attractive as an example of a potentially adaptive application. That characteristic was the application's ability to support both stochastic and deterministic simulation. The Monte Carlo method of simulation is dependent upon the generation of random numbers and the use of probability distributions to simulate real world situations that contain an element of chance. Random number intervals, intended to represent possible outcomes for probabilistic variables in the model, are selected from a random number table or are generated by computer to simulate variable outcomes. The random nature of a stochastic simulation infers that each independent run produces only an estimate of a model's true characteristics for a given set of input parameters. Therefore, a significant number of independent trials must be run in order to evaluate the probabilities of conditions being modeled. On the other hand, a deterministic simulation model, strictly defined, does not contain any probabilistic elements. The outcomes from a deterministic simulation model will not vary from run to run. [KELT91, REND97]

EADSIM is primarily intended to be run stochastically. That is, it is a discrete-event simulation model that employs the Monte Carlo method to simulate the random nature of events in a scenario. A series of such Monte Carlo runs would then be required to adequately evaluate a given scenario. Depending on the complexity and scope of the scenario being simulated, this might require a significant amount of time and computing resources. However, EADSIM also offers the user the opportunity to run a scenario in a deterministic simulation, called the Planner Mode.

EADSIM's Planner Mode is intended to allow the collection of data from a single scenario execution, rather than averaging data over a series of Monte Carlo simulation executions. With the Planner Mode, engaged platforms or ballistic missiles are not allowed to be destroyed; rather, the software accumulates probability of kill (Pk) and engagement geometry data against platforms as the scenario progresses. The C3I process runs without any platforms being killed. A penetrator aircraft is allowed to proceed from a designated beginning time, along its route, to a designated end time, accumulating probability of kill data as a result of weapon engagements, all the way to the designated target. [METH98]

Attrition information is computed and logged for a variety of conditions. This data may be viewed using the Post Processing application, allowing planners to analyze single shot, cumulative, and route probability of kill information for all engagements. Single shot Pk is the Pk for a single shot against a target. Cumulative Pk is the aggregate Pk from all (single) shots taken against a target. Route Pk differs from cumulative Pk in that it includes the effects of weapon reliabilities as well as the attrition of the engaging platform. Consequently, the route Pk is usually less than the cumulative Pk, since single shot Pk will degrade as the attrition of the engaging platform increases. This data may be quite useful in analyzing a strike route, a strike "package," or placement of defensive counter-air assets.

Because no platform is destroyed, post-processing reports resulting from a run in Planner Mode are not readily comparable to post-processing reports generated from Monte Carlo simulations. This lack of similar reports was a challenge to the development of an acceptable measure of effectiveness that can be used to compare the results from the Planner Mode to the results from the Monte Carlo simulation.

It was in examining one possible use of EADSIM that a measure of effectiveness suggested itself. If, it was reasoned, a commander or planner with severe time or resource constraints, was interested in determining the probability that at least one tactical missile (either an air launched cruise missile or a ballistic missile) would reach its intended target, he might want to run the Planner Mode for a quick look, worst case (or best case, depending on the user's view point) estimate. To compute this probability from the Planner Mode (which, it must be stressed, is not the intended use of the application), we deliberately ignored the model's algorithms for determining the effects of engagement dependencies, and assumed all probabilities of kill against missiles were independent events. We then used the product of all final missile route Pk's to determine the probability that every missile was destroyed just before it hit its target. We subtracted this value from 1 to determine the probability that at least one of the missiles was not destroyed:

$$\prod_{i=1}^{n} m_{i\ (\text{final route Pk})} = p(\text{all missiles are destroyed})$$

p(at least one missile is not destroyed) = (1 - p(all missiles are destroyed))
m = probability of kill for the i^{th} tactical missile (cruise missile or TBM)
n = number of missiles in the scenario

This probability could then be compared with the observed outcomes from the Monte Carlo simulation. The MSHN wrappers would measure the difference in computing resources required to run a single Planner Mode simulation and a Monte Carlo simulation (consisting of a statistically significant number of runs).

D. SUMMARY

This chapter discussed the selection of EADSIM as a potential C4I adaptive application. The scenario chosen for experimentation was TBE's Demo300 demonstration scenario, which simulates an air strike on a defender's air base with a leveraging strike from tactical missiles. The scenario was described in section B of this chapter. While EADSIM offers the user a variety of scenario generation, execution and post-processing options, it was important to find a measure of effectiveness with which to compare two versions of the application. The ability of EADSIM to run either stochastic, Monte Carlo method simulations or a deterministic simulation provided an opportunity to choose the probability that at least one missile from the aggressing force strikes its target as the measure of effectiveness.

VI. A DESCRIPTION OF THE EXPERIMENT

This chapter describes the experiment designed to compare the resources required to execute EADSIM running both stochastic and deterministic simulations, to compare the results obtained from these two configurations of EADSIM, and to determine the overhead incurred by the MSHN wrapper. Section A of this chapter describes the computing environment in which the experiment was conducted. Section B discusses the experiment's methodology, explaining how and why each of the three experimental runs was conducted. Section C presents the resource usage data and results obtained from the stochastic simulation, and Section D offers the resource usage data and results from the deterministic simulation. Section E provides the data used to determine the overhead incurred by the MSHN wrapper.

A. HARDWARE AND OPERATING SYSTEMS

1. EADSIM Hardware and Operating System Requirements

EADSIM can either be run on either an SGI or SUN workstation with the minimum requirements noted in Figure 6. 1. It was found that EADSIM would execute with less RAM,

SGI		SUN	
Machine Model:	Indy	Machine Model:	Ultra-1, with Creator 3-D Graphics Card
RAM:	256 MB	RAM:	256 MB
Hard Drive:	1 GB	Hard Drive:	1 GB
Graphics:	24 bit planes, double buffering	Graphics:	24 bit planes, double buffering
OpenGL:	Version 1.1	OpenGL:	Version 1.1
Operating System:	6.x	Operating System:	Solaris 2.5.1 (or higher)
		Window System:	Common Desktop Environment (CDE)* or OpenWindows

*recommended

Figure 6. 1 EADSIM Minimum Hardware and Software Requirements as Stated in EADSIM User's Guide.

but that the C3ISIM GUI executable was not supportable on SUN workstations without the Creator 3-D graphics card.

2. Equipment Used in Experiment

EADSIM Version 7.00 was installed on a Silicon Graphics, Inc. (SGI) IRIX64 server from the CD provided by the U.S. Army Space and Missile Defense Command. All executables and data files, including the demonstration scenarios, were then mounted on an SGI IRIX workstation, a SUN SPARC-20 workstation/server, and a SUN Ultra-1 workstation (Table 2) . Both SGI machines ran the SGI 6.2 operating system, while both SUN workstations ran the Solaris 2.5.1 operating system. These machines were connected by a 10 Megabit per second (Mb/s) Ethernet LAN. The SGI IRIX 64 file server runs SUN's network file service (NFS). Neither the SUN Ultra-1 nor the SUN SPARC-20 workstation were equipped with a Creator 3-D graphics card, which required all pre-and post-processing of the model to be controlled from the command line, vice EADSIM's GUI executable, C3ISIM. The SUN Ultra-1 was upgraded to include 256 MB of RAM, while the SUN SPARC-20 workstation had only 98 MB of RAM.

It was possible to execute the EADSIM run-time processes on both the SUN workstations. Therefore, although EADSIM would execute the simulations on the SUN workstations and output log files and the "stathdr" file needed for post-processing the results of the run, this post-processing of reports could not easily be accomplished on the SUN workstations and, without the 3-D Creator graphics card, there was no means to display the scenario playback graphics on the SUN workstations. The EADSIM GUI was, however, accessible on the SGI workstation (despite the fact that this workstation was equipped with only 32 MB of RAM). By setting the environment path on the SUN workstations to access the scenario data files needed to run the simulation, and setting the resource path on the SGI workstation to access the log files and stathdr file produced by the executables, we were able to run the simulations on the SUN workstations and do post-processing and display the scenario playback graphics on the SGI workstation.

	SGI Server	SGI Workstation	SUN Workstation	SUN Workstation
Type Machine	SGI Challenge L Series	SGI Indy	SUN Sparc-20	SUN Ultra-1
Processor Speed	200 MHZ	100 MHZ	50 MHZ	167 MHZ
Processor Type	MIPS R4400	MIPS R4000	TI,TMS390Z50	SUNW, ULTRASPARC
Number of Processors	4	1	1	1
Amount of Memory	192 Mbytes	32 Mbytes	98 Mbytes	256 Mbytes

Table 2: Configuration of SGI and SUN Workstations Used in Experiment.

B. EXPERIMENT METHODOLOGY

Because the population distributions of EADSIM's resource usage were not known, it was necessary to estimate the mean and standard deviation of resource usage data collected by the MSHN wrapper. It was also necessary to estimate the mean and standard deviation of the cumulative CPU times collected by EADSIM's process statistics `pstat` facility[9] as well as the simulation trial outcomes themselves. The central limit theorem provides a justification for estimating a population mean based on the mean of a large sample of independent observations. The central limit theorem states that the sum of a large number of independent observations from any distribution tends to have a normal distribution [JAIN91]. Using a large sample space (thirty trials), we estimated the population mean of each resource parameter measured, by obtaining the sample mean, standardizing it and using a standard normal table to obtain a confidence interval for the mean. This process also made it possible to estimate the mean of the simulations' results, so that a comparison could be made between the probability of at least one missile reaching its target and the observed sample mean of at least one target being hit.

The purpose of the experiment dictated the method in which it was conducted. The goals of the experiment included the following: (i) use the MSHN wrapper to measure requirements of a pre-determined set of resources (discussed in Chapter IV, Section A) for each of the three EADSIM executables running the stochastic simulation (employing Monte Carlo methods); (ii) use the MSHN wrapper to measure the computing requirements of the same resources for each of the three EADSIM executables running the deterministic simulation (Planner Mode); (iii) measure the overhead incurred by the MSHN wrapper by using the EADSIM pstat facility to measure the cumulative CPU time of both the wrapped and unwrapped executables running the deterministic simulation; and (iv) use EADSIM's post-processing report generation to compare results from the stochastic simulation with results form the deterministic simulation. Each of these goals suggested a specific method of gathering data.

An observation that was made early in the experiment design process, was that the number of page faults and the related size of the resident set in memory were directly

[9] EADSIM's `pstat` functionality, designed into each of the run-time process executables, makes periodic system calls using the shell built-in function, `times` (prints accumulated process times for user and system), while its process is executing. Cumulative CPU time (as well as accumulated data regarding wall clock time and memory used by the process) is then output to a `pstat` file for each run-time process. Use of the `ps` command in the MSHN wrapper to compute user CPU time and system CPU time is discussed in Chapter IV, Section B.

correlated to wall clock time. It became clear that, in order to provide as controlled an environment as possible in measuring resource usage, the number of page faults would have to be standardized for each trial run. The only way to ensure this was to run each trial on a dedicated workstation, and at a time when there would be as little competition for memory as possible, thereby ensuring no page faults. Simulations were run consecutively, after normal working hours, on the SUN Ultra-1 workstation, and each trial was checked to ensure the absence of page faults. In reality, it is anticipated that EADSIM would be used by the warfighter in a "reach-back" situation. That is, the commander or planner would be forward deployed in a possibly computer-resource-poor environment, and would access EADSIM, located at a meta-computing center, via a reach-back network. Without the use of an RMS like MSHN, it is unlikely that the commander would know whether the server he was accessing was being shared by other applications or what impact this would have on his expected run time. It is sufficient to note, that wall clock time will increase if an application is required to swap more pages in and out of memory during execution.

Using the Demo300 scenario, three runs of thirty trials each were conducted to gather the data needed to meet the experiment's goals. The first run consisted of thirty Monte Carlo simulation trials executed by the wrapped version of each of the three EADSIM run-time processes (C3I, Detect, and FP). A run script (sh script) was written that allowed the seed for the model's random number generator to be tied to the system clock (APPENDIX E). Both the C3I process and the Detect process were given different seeds for each trial (the FP process does not require a random number generator). Data collected from these thirty Monte Carlo simulation trials included MSHN wrapper output for each resource measured, cumulative CPU time as measured by EADSIM's `pstat` facility, and post-processing reports that had been tailored to gather data on missile success or failure. The second run consisted of thirty deterministic (Planner Mode) simulation trials executed by the wrapped version of each of EADSIM's three run-time processes. A separate run script similar to that used to launch the Monte Carlo simulation was employed, with no seed applied. Data collected from these thirty deterministic simulation trials included MSHN wrapper output for each parameter measured, cumulative CPU time as measured by EADSIM's `pstat` facility, and a post-processing report (the PAPA1 report) that provided accumulated Pk data for each missile launched. Of note, only one such report was needed since simulation outcomes were deterministic. The final run consisted of thirty deterministic (Planner Mode) simulation trials executed by the unwrapped version of each of EADSIM's three run-time processes. Data collected from these trials consisted only of cumulative CPU time as measured by EADSIM's pstat facility.

The data collected from the experiment trials were gathered into separate directories by another sh script, and copied into text files by PERL scripts written for this purpose. These text files were then copied into a statistical analysis computer program called Minitab which was used to produce descriptive statistics for each parameter measured. Minitab then produced histograms (see APPENDICES H and I) from these statistics and plotted these histograms against normal curves for graphical comparison. As stated in Chapter I, Section C, follow on research will be conducted in an attempt to determine distributions for each of the resource usage parameters monitored by the MSHN wrapper [COOK99].

C. RESULTS FROM WRAPPED STOCHASTIC RUNS

As discussed in Section B above, after executing preliminary runs to ensure required pages were cached in memory and no page faults were recorded, thirty Monte Carlo simulation trials of the Demo300 scenario were run using the three wrapped, EADSIM executables (C3I, Detect, FP). Output from the wrapper, the `pstat` facility and the post-processing reports was collected and stored in a dedicated directory. Data was sorted and, when appropriate, copied into Minitab for further analysis.

1. Resource Measurements

Computing resource usage data, collected using the MSHN wrapper during the thirty Monte Carlo simulation trials described in Section B, is summarized in Table 3. Graphic representations of statistics produced from those values that demonstrate variance during the Monte Carlo thirty trials are available in APPENDIX H. Values that did not change from one trial to the next are highlighted and noted with an asterisk. In the FP executable, only once in the thirty Monte Carlo trials did the number of pages in the resident set vary (one trial had 7904 pages, vice the 7912 seen in the other Monte Carlo runs). This value is both highlighted and marked with a double asterisk. Since network reads (client machine seeking data from the network file server) were only necessary for process initialization, it was to be expected that the number of bytes read across the network, and the number of network reads, remained constant from one trial to the next for each process. Likewise terminal I/O, which was limited to output to the screen during the initialization process, did not vary from trial to trial. The number of data bytes read and the number of reads done locally (Local File Data) did not vary by process or from run to run (there were no local data writes recorded). This data was probably read from the local `proc/` directory (related to the `pstat` functionality of each process). With the exception noted above, the number of pages in physical memory, virtual memory and resident set did not change from trial to trial in the FP and Detect processes, but did vary slightly from trial to trial in the C3I process (standard deviation over the thirty trials was minimal, as can be seen in APPENDIX H).

	C3I Process	FP Process	Detect Process
Local IPC: Number of Bytes Read	166758	3782	78411
Local IPC: Number of Reads	20845	473	9801
Local IPC: Number of Bytes Written	69328	2244479	1358588
Local IPC: Number of Writes	1077	59372	640*
Local File Data: Number of Bytes Read	2945	2945	2945
Local File Data: Number of Reads	5	5	5
Network (NFS): Number of Bytes Read	708605*	708605*	704678*
Network (NFS): Number of Reads	129*	129*	124*
Network (NFS): Number of Bytes Written	1634463	1029378	2057529
Network (NFS): Number of Writes	155957	741	589
Terminal I/O: Bytes Written	5377*	530*	307*
Terminal I/O: Number of Writes	[illegible]	21*	12*
System CPU Time	3.03	3.20	5.86
User CPU Time	17.72	17.13	16.32
Cumulative CPU Time (measured by EADSIM pstat)	20.73	20.31	22.15
Wall Clock Time	94.52	77.17	93.36
Page Faults	[illegible]	[illegible]	0*
Nmber of Physical Pages in Memory	2953	1281*	1294*
Number of Pages in Virtual Memory	23628	10248*	10360*
Number of Pages in Resident Set	17965	7912**	7752*

* Outcomes did not change from trial to trial
** Outcome of only one trial differed from the other 29

Table 3: Mean Resource Usage, Over 30 Monte Carlo Trials, As Measured by MSHN Wrapper.

2. Observed Simulation Outcomes

Using the C3ISIM GUI interface on the SGI Indy workstation, we were able to tailor several post-processing reports to analyze the outputs from the stochastic runs of the Demo300 scenario. Having selected the observed number of times that at least one missile reached its target as our measure of effectiveness (Chapter V, Section C discusses the measure of effectiveness in detail), we tailored an Action History report to produce Red successes only (Figure 6.2). By simply counting the number of successes against the missiles targets, as defined by the Demo300 scenario, we were able to determine the observed percentage of at least one missile reaching its target over the thirty Monte Carlo runs. Since there were seven occasions, in thirty trials, when at least one missile successfully destroyed its target, the proportion observed was 0.23.

```
!                  Engagement Statistics
!                  --------------------
!
!  Scenario: Demo300
!
!  Report Type: Action History
!
!  Begin Report Time: 0
!  End Report Time: 960
!
!  Actions:
!  --------
! Hit_Base
! Success
!
!  Acting                     Against
!  Platforms:                 Platforms:
!  ----------               ----------
! All Hostile                All Friendly
! All Missiles               All Ground Units
!                            All Air Units
!                            All Missiles
!
!            Acting                                Against
! Time      Platforms               Action          Platforms
! ------ -------------------- ------------------------ -------------------
-
239.00 TBM_Depr_Traj_02   Hit_Base          BASE
246.42 Hostile_AA_Ftr_02/01 Success          AA_Ftr_04/02
294.50 Hostile_AA_Ftr_01/04 Success          AA_Ftr_05/02
301.02 Hostile_AA_Ftr_03/01 Success          AA_Ftr_04/01

Total Number of Actions : 4
```

Figure 6. 2 Sample Red Action History Report Summarizing All Red Missile Hits and Air-to-Air Combat Successes Against Blue Assets. After [EADS98]

D. RESULTS FROM WRAPPED DETERMINISTIC RUNS

1. Resource Measurements

Resource usage data collected by the MSHN wrapper during the thirty, deterministic (Planner Mode) trials is summarized in Table 4. Since outcomes from a deterministic model do not vary from run to run, the only resource usage data that produced a non-constant distribution across the thirty trials were user CPU time, system CPU time, and wall clock time. The resource usage values that did not vary from those observed during the Monte Carlo trials are highlighted in Table 4.

As expected, since the Planner Mode allows each missile to reach its intended target, and accumulates Pk data rather than allowing an engagement to result in a platform being destroyed, each trial in the Planner Mode required more resources than trials run with the stochastic model. There was substantially more CPU usage and wall clock run-time in the Planner Mode, as well as more bytes written to disk, number of writes to disk, number of bytes read and written via interprocess communication, and number of IPC reads and writes. In the deterministic simulation (Planner Mode) only wall clock time, and system and user CPU (cumulative) time varied from one run to another. However, none of these increases would cause the expected run-time of a single deterministic run to be equivalent to thirty Monte Carlo trials. Graphical representations of statistics produced from these values that varied during the thirty deterministic trials is available in APPENDIX I.

	C3I Process	FP Process	Detect Process
Local IPC: Number of Bytes Read	223784	4600	111768
Local IPC: Number of Reads	27973	575	13971
Local IPC: Number of Bytes Written	119840	3098268	2497276
Local IPC: Number of Writes	1347	81968	640*
Local File Data: Number of Bytes Read	2945*	2945*	2945*
Local File Data: Number of Reads	5*	5*	5*
Network (NFS): Number of Bytes Read	708605*	708605*	704678*
Network (NFS): Number of Reads	129*	129*	124*
Network (NFS): Number of Bytes Written	2892142	1381809	2612821
Network (NFS): Number of Writes	290730	753	645
Terminal I/O: Bytes Written	5377*	530*	607*
Terminal I/O: Number of Writes	111*	21*	12*
System CPU Time	5.07	4.36	11.20
User CPU Time	34.36	20.54	27.12
Cumulative CPU Time (measured by EADSIM pstat)	39.40	24.88	38.30
Wall Clock Time	141.02	124.50	140.81
Page Faults	0*	0*	0*
Number of Physical Pages in Memory	2965	1281*	1294*
Number of Pages in Virtual Memory	23720	10248*	10360*
Number of Pages in Resident Set	18040	7904	7752*

*** Outcomes did not vary from those observed in Monte Carlo trials.**

Table 4: Mean Resource Usage for 30 Deterministic Trials, as Measured by MSHN Wrapper.

2. Computed Outcome Probabilities

As discussed in Section B of this chapter, EADSIM's post-processing GUI offers a series of reports useful for analyzing scenarios run in the Planner Mode. These reports are called "PAPA" reports, an important one of which is known as the PAPA1 report. Although the PAPA1 report provides analysts with a variety of information regarding hostile and friendly platforms, as discussed in Chapter V, Section C, we are most interested in the route probability of kill (route Pk) for each tactical missile (TBM and cruise) launched. We can obtain this data by scanning the PAPA1 report for the last given route Pk for each missile. APPENDIX J provides a PAPA1 report, with final route Pk's of each missile highlighted.

In Chapter V, Section C it was determined that our measure of effectiveness would require us to determine the probability that at least one missile reached its target (it is assumed in our experiment that if a missile is not destroyed, it will hit its intended target), in order to compare this probability with the outcomes observed in the Monte Carlo trials. Recall that the equation to determine whether at least one missile reached its target was:

$$\prod_{i=1}^{n} m_{i\ (\text{final route Pk})} = p(\text{all missiles are destroyed})$$

p(at least one missile is not destroyed) = (1 - p(all missiles are destroyed))
m = probability of kill for the ith tactical missile (cruise missile or TBM)
n = number of missiles in the scenario

From this PAPA1 report, we find that there were a total of twenty-one TBMs and four cruise missiles launched, for a total of twenty-five tactical missiles. Taking the product of each missile's final route Pk (highlighted in APPENDIX J), we find that the probability that all missiles were destroyed before hitting their target is 0.289. Subtracting this number from one, we conclude that the probability that at least one missile was not destroyed (and hence hit its target) is 0.711.

E. MEASURING THE OVERHEAD OF THE WRAPPERS

In order to determine the overhead incurred by the MSHN wrapper, we ran thirty deterministic simulation trials of the Demo300 scenario using the three wrapped executable processes (C3I, FP, and Detect) and thirty trials using unwrapped processes. Using output from the `pstat` function discussed in Section B of this chapter, we found the mean cumulative CPU time of each process for each set of thirty trials. By comparing the mean cumulative CPU time gathered from the trials executed by the wrapped processes with the cumulative mean gathered from the trials executed by the unwrapped processes, we were able to determine the difference in average cumulative time for each process (Table 5). Since all trials were deterministic and run on the same workstation with the same file server, we attributed this difference to the overhead incurred by the wrapper.

PSTAT DATA (in seconds)	Cumulative CPU Time Wrapped	Cumulative CPU Time Unwrapped	Difference
C3I Process	39.39	38.51	0.88
FP Process	24.88	24.73	0.15
Detect Process	38.30	29.21	9.09

Table 5: Mean Cumulative CPU Times Reported by PSTAT Function (Planner Mode).

It is worth noting that the cumulative CPU times for the wrapped processes produced by the `pstat` function called by EADSIM closely agree with the cumulative CPU times measured by the MSHN wrapper. The mean cumulative CPU times (mean user CPU + mean system CPU) for the thirty deterministic simulation trials executed by the wrapped processes, as measured by the MSHN wrapper, are given in Table 6. In each case, the mean cumulative CPU time derived from data gathered by the MSHN wrapper was within approximately 1% of the mean cumulative CPU time derived from the `pstat` reports. As is evident from Table 5, only the Detect process incurred significant overhead (31%) from the MSHN wrapper. A graphical representation of the statistics derived from the wrapped and unwrapped executables running the Demo300 scenario in deterministic simulation trials, as measured by the `pstat` functionality of each process, is provided in APPENDIX K.

MSHN Wrapper Data (in seconds)	Cumulative CPU Time
C3I Process	39.43
FP Process	24.90
Detect Process	38.32

Table 6: Mean Cumulative CPU Times Reported by MSHN Wrapper.

Since it was anticipated that the MSHN wrapper would add an insignificant amount of overhead to each wrapped process, it was surprising that the wrapped version of the Detect process demonstrated a nine second increase in cumulative CPU time over the cumulative CPU time measured for the unwrapped version. To attempt to determine the cause of this increase we took an algebraic approach that compared wrapper invocation overhead for the deterministic simulation trials. We assumed that any overhead added by the MSHN wrapper depended upon the actions taken by the wrapper upon intercepting a system call. The first step of this evaluation was to determine which wrapper calls did not vary from process to process, and which could not have contributed to the overhead incurred by the wrapper. Referring to Table 4, we noted that local file data and the number of page faults did not vary from process to process, and could therefore be eliminated as a possible source for the increased overhead displayed by the Detect process. Upon examining the code in APPENDIX C, we saw that the amount of data (number of bytes) transferred after a system call has been intercepted does not contribute to the overhead incurred by the wrapper. In other words, while the number of `read()`, `write()`, `open()`, `close()`, `connect()`, and `accept()` operations was of interest in evaluating the overhead accrued by the wrapper, the number of bytes transferred during, or as a result of, each of those operations would not have contributed to wrapper-generated overhead. By the same token, the number of pages in physical memory would not have contributed to the overhead created by the wrapper, since the wrapper simply gathers the data which is does by making a single, per process, invocation of the `ps` facility (Chapter IV, Section B discusses use of the Unix `ps` command, and code for the `getPsData()` function is available in APPENDIX D).

One final observation needs to be made before continuing our analysis. Terminal I/O debug statements, which were inserted in the MSHN wrapper source code during the debug process[10], were left in the code throughout each of the three, thirty-trial runs. When the experiment was completed, we removed the debug statements and executed the Demo300 scenario using the wrapped executables running the deterministic (Planner Mode) simulation. We found that all twelve of the terminal I/O writes for the Detect process noted in Table 4 had been eliminated, as had five of the one hundred and eleven terminal I/O writes performed by the C3I process and thirteen of the twenty-one terminal I/O writes performed by the FP process. While this would explain part of the total overhead incurred by the Detect process, since the FP process had more terminal I/O attributed to the debug statements than did the Detect process, we concluded that the debug statements can also be eliminated as a source of the added overhead observed in the Detect process.

In comparing the values measured for each of the three processes, one value stands out as being significantly different for the Detect process than for either the C3I process or the FP process. The ratio of Detect's system CPU time to its cumulative CPU time is significantly greater than either of the other two processes' ratios of system CPU time to cumulative CPU time. This was observed in both the deterministic simulation trials and the stochastic simulation trials. Perhaps more interesting, however, is the fact that although the Detect process required far less user CPU time than the C3I process to execute the Demo300 scenario in the deterministic simulation, the Detect process required more than twice as much system CPU time than did the C3I process (which was, in turn, more system CPU time than was required by the FP process). This seemed to be an excellent clue for evaluating why the MSHN wrapper incurred more overhead in the Detect process than in the other two processes. Something within the Detect source code was requiring that more time be spent in system calls than in the other two processes. We decided to compare the quantity of each different type of system call that the wrapped process made, in an attempt to determine which, if any, of Detect's system calls were significantly greater in number than in the other two wrapped processes.

[10] #Ifdef MSHN DEBUG statements written into many of the MSHN client library functions were not defined for use during this research. Debug statements were instead inserted into four functions only.

Again referring to Table 4, we concentrated our effort on the following values: number of local IPC reads and writes; number of local file data reads; number of remote file data read from and written to the network file server; and the number of terminal I/O writes. As can be seen by scanning Table 4, in no case did the Detect processes' resource usage measured for any of these values exceed the usage measured for each of the other two processes (the number of reads and writes listed in the Detect Process column of Table 4, is never the maximum value of a given row). In all but one case (the number of IPC reads) both the C3I process and the FP process performed at least as many read and write operations as the Detect process. And while the mean number of IPC reads is greater for the Detect process than the FP process, this number is far less than the mean of observed IPC reads for the C3I process.

Based on the fact that the Detect process did not perform the most read or write system calls, of any given type (local IPC, local file data, remote file data, and terminal I/O), we eliminated wrappers of read and write system calls as the cause of the significant wrapper-induced overhead observed in the Detect process. However, as discussed above, we know that the Detect process did require significantly more system CPU time than the other two processes. To understand how this can be possible we reviewed the way in which EADSIM models a scenario (EADSIM is described in detail in Chapters III and V and the Demo300 scenario is described in Chapter V).

Through IPC, the C3I process tells the Detect process which platforms are of interest. The Detect process, which has been initialized with sensor characteristic data for each of the scenario's platforms, receives "ground truth" on the movement of these platforms via IPC from the FP process. The Detect process uses this data to compute sensor output, then periodically reports sensor information to the C3I process. While EADSIM's source code was not used in this analysis, it was verified by TBE engineers that the Detect process awaits ground truth from the FP process, in order to make its periodic report of detections to the C3I process. We believe that during each time-step interval, the Detect process queries the FP process, in a loop, until ground truth data is sent. This is known as "busy waiting." In this busy waiting situation, the Detect process would use the CPU to check (in a loop) whether the FP process had data to transmit. The MSHN wrapper does cause some overhead each time the Detect process checks the FP socket for data. Even when no data is available in the socket, the wrapper calls `getclocktime( )` twice. To eliminate the possibility that the MSHN wrapper was adding overhead by invoking the `getclocktime( )` system call each time the Detect process checked for data from the FP process while busy waiting, the call to `getclocktime( )` was commented out of the `readWrapper( )` function in MSHN_syscall_lib.cc. This change, however, resulted in no discernable reduction in system, or cumulative, CPU time required to execute the Detect process.

The MSHN wrapper does not currently report very fine grained data regarding system calls. In fact, no data is output regarding the number of `open( )`, `close( )`, `connect( )`, or `accept( )` system calls. Therefore, having eliminated the data that the MSHN wrapper output to file as providing sufficient information from which to deduce the cause of the additional overhead incurred by the Detect process, we conclude that we currently have too little data to determine what caused the MSHN wrapper of the Detect process to incur more overhead than the wrappers of the other two processes. The solution to this problem will be discussed in the Future Work section of the next chapter.

One last note regarding the data output to file by the MSHN wrapper. It is apparent in Table 4 that the total number of IPC writes do not equal the total number of IPC reads, nor do the total number of IPC bytes written equal the total number of IPC bytes read. It is unclear, based on the data available from the MSHN wrapper, why this is so. Work is currently underway to modify the MSHN wrappers so that finer grained data will be collected and reported.

F. WEIGHING THE SIMULATION RESULTS USING OUR MEASURE OF EFFECTIVENESS

As discussed in Chapter V, Section A, we have chosen EADSIM as being representative of a complex C4I modeling application that offers the warfighter a variety in Quality of Service. Based on the operational situation, the commander or mission planner, with the aid of an intelligent resource management system such as MSHN, can select that version of EADSIM that best suits his or her needs. This decision will take into consideration the time constraints, system security limitations, mission priority, and computing resource availability (e.g., network bandwidth and anticipated congestion, local memory and disk space, graphic display capabilities). Chapter V, Section C, explains a measure of effectiveness that would allow us to weigh the results obtained from running EADSIM in a deterministic simulation with the results obtained from running EADSIM in a stochastic simulation. In other words, in a time constrained situation, given that the deterministic simulation, though more resource intensive than the stochastic simulation, need only be run once, can the results obtained from the deterministic version of EADSIM be useful? Or must mission planning be delayed sufficiently long to enable the Monte Carlo runs to be executed a large number of times?

Table 7 displays the computed probability (from the Planner Mode run) that at least one missile would not be destroyed (and would hit its target) and the observed percentage of time (from the Monte Carlo runs) that at least one missile hit its target. What, if any, conclusions can be drawn from this data?

Using data from the PAPA1 report described in section D, subsection 2, and our equation to determine the probability that at least one missile hits its target, the commander who chose to run the deterministic simulation due to time constraints, would have been told there was a 71% chance, one of his or her bases would be hit by a missile. Had the same commander, with more time available to analyze the scenario, chosen to run thirty stochastic trials, he or she would have observed one of his or her bases hit by a missile in only 23 % of the runs. Although the deterministic simulation offers a significant savings in time, there was quite a disparity between the probability of a missile hit, and the observed proportion of times at least one missile hit its target.

	p(at least one missile hits its target)
Deterministic Simulation	.711
Stochastic Simulation	.233
	Proportion of trials when at least one missile hit target

Table 7: Comparison of Predicted and Observed Outcomes.

To determine whether the observed disparity could simply be due to chance, or the overall probability of a missile reaching its target in the stochastic trials was, if fact, different than that for the deterministic trials, we next tested the hypothesis that the true underlying probability of at least one missile hitting its target during the stochastic trials was $p = .711$. Each run of the stochastic simulation can be considered a Bernoulli trial, with the probability of at least one missile hitting its target equal to p, and the probability of no missile hitting its target equal to $1-p$. The number of success (trials in which at least one missile hits its target) of n repeated independent Bernoulli trials (each with probability p) follows a Binomial distribution with parameters (n and p). In our case, the observed number of successes (in $n = 30$ trials) was 7. So, the objective of the test was to determine whether 7 successes in 30 trials was a reasonable outcome if the true probability of success in each trial was .711.

Figure 6. 3 provides the following: (i) the Binomial distribution and values for n, x, and p; (ii) the null hypothesis, that the probability of at least one missile hitting its target is .711; (iii) the observed proportion of successes (at least one missile reaching its target) in thirty Monte Carlo trials; and (iv) the probability that an observed proportion of successes would be less than or equal to .233, given the probability of at least one missile hitting its target is .711.

(i) Binomial Distribution: $$b(x; n, p) = \begin{cases} \binom{n}{x} p^x (1-p)^{n-x} & x = 0, 1, \ldots, n \\ 0 & \text{otherwise} \end{cases}$$

Where:

n = number of trials = **30**

x = number of successes = **7**

p = probability of success = .711

(ii) Null hypothesis (H_0): p(at least one missile reaches its target) = .711

(iii) $\hat{p}$ = proportion of x/n = 7/30 = .233

(iv) $p(\hat{p} \leq .233 \mid p = .711) = 0.0000$

Figure 6. 3 Testing the Null Hypothesis that the Probability of at Least One Missile Reaching Its Target is .711.

As Figure 6. 3 part (iv) shows, if the true probability of at least one missile reaching its target is .711 (our null hypothesis), then the probability we would observe a result as unusual as x = 7 or less (the p-value), is essentially 0.0. Therefore we can reject the null hypothesis with any reasonable level of significance. In other words, applying a two-tailed hypothesis-test to determine whether our observed proportion of successes (success is defined as at least one missile reaching its target) supports our null hypothesis with an acceptable level of confidence, would lead us to reject the null hypothesis. By rejecting the null hypothesis, we have concluded that the overall probability of at least one missile reaching its target in thirty Monte Carlo trials is not equal to the .711 derived from the deterministic simulation. While rejection of the null hypothesis means that we cannot simply substitute one version of the model for the other, it does not infer that the deterministic version of EADSIM cannot be of use if time constraints (the operational deadline) eliminate the possibility of running the stochastic version. It would be very helpful, however, to know which direction the disparity might go, and to have an estimate of the bias.

The defending commander, using EADSIM, who chose to go with the deterministic version of the model, would know that he or she had received a worst case predicted loss. Planning based on this anticipated result, may have been far more conservative than necessary, but in wartime this may not be a bad alternative. As long as the commander was apprised of the potential for gross over-estimation of the enemy's capabilities, the result from the deterministic simulation may represent a "better than nothing" estimate. Furthermore, it must be remembered, that the PAPA1 report (APPENDIX J) is a valuable resource for analysts, providing a means to follow scripted platforms from launch to intended target, accumulating probability of kill and flight geometry data along the route. The defending commander for instance, might concentrate more defensive counter air in the area of the Red Cruise Missile Number 2, which, according to the PAPA1 report, had only a 70% chance of being destroyed before it reached its target (vice over 97%, for all other cruise missiles).

On the other hand, had the offensive planners using EADSIM decided they only had time for the deterministic version of the model, they would know that they had received an overly optimistic probability that at least one of their missiles hits its target.

But the computed probability of at least one missile hitting its target was just the measure of effectiveness chosen for this experiment. The PAPA1 report would have provided the offensive planners with valuable data regarding the relative probability of kill for each of their platforms for each leg of the route to their intended targets. This information might be used to evaluate strike packages, flight routes, target accessibility, sensor requirements, or placement of ground assets. In some after-action analysis, contingency and mission planning, data gathered from running scenarios in the Planner Mode (deterministic simulation) may be just as valuable as running stochastic simulations for observed outcomes. It must be stressed that QoS is determined by the user, in our case the warfighter, who will choose the version of an application that most closely suits the needs and constraints of the given operational situation, based upon advice from an RMS.

G. SUMMARY

This chapter described the experiment that allowed us to compare the resources required to execute EADSIM running both stochastic and deterministic simulations, to compare the results obtained from these two configurations of EADSIM, and to determine the overhead incurred by the MSHN wrapper. It provided the suggested minimum hardware requirements to run the EADSIM application and described the computing environment in which our experiment was conducted. It detailed the methodology we used to obtain data in support of the experiment's objectives. It reported the resource usage derived from the thirty stochastic simulation trials and the observed simulation outcomes and the resource usage from the thirty, deterministic simulation trials that were run using the wrapped executables. It analyzed, as best as possible, the observed overhead attributed to the MSHN wrapper. Finally this chapter weighed the results obtained from the deterministic simulation against the results obtained from the stochastic simulation.

VII. CONCLUSIONS AND FUTURE WORK

This thesis presented the methodology of intercepting, or wrapping, system calls made by the Extended Air Defense Simulation (EADSIM), a robust C4I, air and missile warfare modeling application, in order to determine the resources required to execute the program on a stand-alone workstation. Having demonstrated the ability to measure the application's resource usage without requiring access to source code, an experiment was described in which the resource usage was measured running the application in both Monte Carlo simulations and deterministic simulations. The outcomes obtained from running EADSIM in both deterministic and stochastic simulations were then weighed against each other. This chapter will discuss the contributions of this thesis and future work.

A. CONCLUSION

The goal of MSHN is to provide a resource management system (RMS) that will enable adaptive applications to provide each subscriber process with its required Quality of Service (which might include security considerations, deadlines, user priorities, and preferences) in a distributed, heterogeneous computing environment in which many processes are competing for shared resources. The MSHN architecture is designed as a system that is capable of being integrated with, and incorporating, a variety of distributed system tools (for example, CORBA, ENSEMBLE, and COMPASS) to reap the maximum benefits from available resources.

In a military environment, where the use of distributed, and possibly heterogeneous, systems represents both challenge and opportunity, the MSHN RMS would allow a user to select the most appropriate application, or version of an application, capable of executing within a specified time, at the proper security level, in order to deliver the most complete answer achievable within stated time constraints. Applying this technology to C4I modeling and simulation applications would enable on-scene commanders and mission planners to simulate complex elements of the decision process in order to optimize the use of forces and materiel. As discussed in Chapter II, critical to this implementation is the development of adaptive and adaptation-aware models and decision support applications that exist in different versions, designed to satisfy user-defined Quality of Service (QoS) parameters.

It was explained in Chapter II that adaptive applications exist in different versions capable of producing like results (though possibly offering varying degrees of QoS). MSHN would monitor the use of such adaptive applications and would be able to terminate one version and start another, possibly from the beginning, if it perceived the user's QoS requirements were not being met by the currently executing version. In a tactical environment, this means that the transition to improved QoS recommended by the MSHN RMS, if accepted by the decision-maker, would transparently enhance his or her mission effectiveness while remaining within given time constraints.

Chapter IV described the experiment that we conducted to weigh model results against the resources required to execute deterministic and stochastic model simulations. This experiment demonstrated MSHN's ability to measure an application's resource usage without requiring source code. The overhead associated with the MSHN wrapper was measured and, in Chapter VI, possible causes of this overhead were discussed. The experiment led to the realization that the MSHN wrapper needs to be modified to collect finer grained information, specifically, `send ( )`, `sendto ( )`, `sendmsg ( )`, `recv ( )`, `recvfrom ( )`, `recvmsg ( )`, `select ( )`, and `listen ( )` system calls may need to be wrapped. Additionally, more information may be required from the current wrapper. Chapter VI also used the Binomial distribution to help evaluate the trade-off between the fidelity of results from EADSIM, a sophisticated C4I simulation.

B. FUTURE WORK

1. Development of a MSHN Application Emulator

Resource usage data gathered by the MSHN wrapper will be used for, among other things, the MSHN Application Emulator (AE). The AE will produce predictive statistics regarding resource requirements by simulating the running of applications without the accompanying overhead. The AE will also be used to monitor the status of resources not being used by MSHN applications. In order to simulate an application, the AE must know that application's resource requirements and the probability distribution(s) of those requirements. Data from applications wrapped by the MSHN Client Library, similar to the data reported for EADSIM resource usage, will be used in the MSHN AE, along with their appropriate distributions, to simulate applications being executed in a distributed environment. [DRAK99]

2. Dynamically Determining Distribution Statistics for Resources in a Distributed Environment

Statistical analysis needs to be conducted, using resource usage data collected by the MSHN wrapper, to determine whether there is a family of distributions for a given resource requirement (similar to the Student's t family of distributions) that can be adequately modeled by an underlying exponential distribution. Assuming such an exponential family of distributions exists, Monte Carlo simulations will be necessary to compare the performance of greedy and exhaustive scheduling algorithms using the exponential family of distributions with the performance of these same scheduling algorithms using a normal distribution. Research will also have to determine what sample size is needed for the resultant distribution to be within an acceptable level of significance of the true exponential distribution. [COOK99]

3. Refining a Model for Use in Scheduling in MSHN

In order to accurately predict an application's performance, in a given distributed environment, a network system model will need to be developed. This model will use the MSHN Application Emulator to emulate computationally-intensive and communication-intensive, synchronous and asynchronous, applications. Using data generated by the MSHN wrapper and the Application Emulator, the network system model will focus on predicting run-times for communication-intensive and computationally-intensive processes. Since MSHN's goal is to provide a resource management system (RMS) that will provide each subscriber process, when possible, with its required Quality of Service in a distributed, heterogeneous computing environment, this information is needed to determine whether next-generation C4I requirements (among others) can be supported by Commercial-Off-the-Shelf and Government-Off-the-Shelf products. [SHAE00]

4. Testing Resource Monitoring Tools on a Win32/Intel Platform

As part of Information Technology, 21st century (IT21) the US Navy has made a commitment to transition from UNIX workstations to a Windows-based, NT computing environment. Methods of monitoring resource usage for applications being executed on Win32x86, or NT, platforms are needed. Such monitoring will parallel the methods used by the MSHN wrapper in the UNIX computing environment. In other words, a method of measuring resource usage for an application run on NT workstations and servers will be sought that does not require access to the application's source code. It is anticipated that this investigation will include work done with the Executable Editing Library (EEL) developed by James Larus of the University of Wisconsin, and an extension of EEL for the Win32 Platform developed by the Etch team from the University of Washington. [CHEN00]

5. Expansion of Existing MSHN Wrapper Functionality

As discussed in Chapter VI, the MSHN wrapper will need to be expanded to capture more fine grained data than is currently being measured. This will mean that several more system calls will need to be intercepted, allowing more resource data to be analyzed. System calls that need to be wrapped include `send ( )`, `sendto ( )`, `sendmsg ( )`, `recv ( )`, `recvfrom ( )`, `recvmsg ( )`, `select ( )`, and `listen ( )`. Once these system calls have been wrapped by MSHN, further experiments can be run on EADSIM and other military applications, to better understand resource requirements, and to facilitate the research discussed in Section B, Subsections 1-4 above.

APPENDIX A. MODIFIED README-FIRST FILE

This README FIRST file was written by Matthew Schnaidt for use with the MSHN Client Library files [SCHN98]. I have modified this file for use with the Solaris 2.5 (Sun5) operating system (changes noted in bold).

```
FILE:  README-FIRST

1. PURPOSE.  The purpose of this readme file is to assist the user in
using the MSHN client library.  A certain level of understanding of the
client library is assumed.  Matt Schnaidt's December '98 thesis is the
basis for this knowledge.  Specifically, Chapters 2, 4, 5, 6, 7 and
Appendices B and C.  The author strongly recommends working through the
tutorial in Appendix C prior to trying to run link with the client
library.

2. SUBDIRECTORIES.  The subdirectories in this directory contain files
used by the client library as well as test programs used with the
client library.  The subdirectories and the purpose of each are
enumerated below.

      /syscall_lib      This directory contains the files required to
wrap system calls.

      /extract          This directory contains the makefile that
creates the modified C library used by the syscall library wrappers.

      /makeUppercase    This directory contains the code for the
makeUppercase.cc and makeLowercase.cc programs.

      /clock            This directory contains the code for the
clockServer used in estimating clock offsets.

      /thesisTutorial   This directory contains the code presented in
Appendix C of Matt Schnaidt's thesis.  This presents a simple tutorial
on how to wrap system calls.

      /socketTest       This directory contains a client and server
program.  These programs are wrapped with the client library.  A server
is started on one machine and a client is started on another; these
programs simply transfer a sample file back and forth in order to
demonstrate the client library's functionality.

      /testoverhead     This directory contains code for testing the
overhead incurred by the client library.

3. RUNNING A WRAPPED PROGRAM.  Each subdirectory has a README file that
explains how to use it.  In order to wrap an application, the following
is a high level description of tasks that must be accomplished:

a. Create the modified C library, libMSHNc.a.  Enter the /extract
directory, and run the mshnlibc*Makefile (where * is the current OS
name - Linux, Sun4 etc) (e.g., make -f mshnlibcSun4Makefile).  This
```

will create the modified C library, libMSHNc.a. Copy this library into the /syscall_lib directory.

b. Create the client library, MSHN_syscall_lib.o. Enter the /syscall_lib directory (ensure that you've copied libMSHNc.a to this directory). Check the file MSHN_types.h to ensure that the correct variables are defined (e.g., LINUX, SUN4, SUN5), and then compile using the makefile, libraryMakefile(e.g., make -f libraryMakefile). At this point, you now have the client library, MSHN_syscall_lib.o.

c. Modify the C Run-Time object file. Enter the extract directory. Copy the C run-time object file to this directory and rename it Mod_crt*.o (on Sun4 this is crt0.o, so cp /usr/lib/crt0.o ./Mod_crt0.o; **in Solaris 2.5 this is crt1.o, so cp /usr/lib/crt0.o ./Mod_crt1.o**). Now, modify the C run-time object file's reference to main() (makeUppercase Mod_crt0.o main **or Mod_crt1.o main**). In order to link any applications with this one, you will link with MSHN_syscall_lib.o. Steps a, b,and c need not be repeated.

d. Link your application with the client library and modified C run-time object file. First, copy the MSHN_syscall_lib.o and Mod_crt0.o **(or Mod_crt1.o)** files into this directory. The simplest way to do this is to compile your application into a single, non-executable, object file (e.g., myApplication.o). Then linking will take place in two phases: preparation, and linking. We are going to modify the link command generated by the compiler. So, the first step is to determine the link command that the compiler uses, and to modify it to replace the system's C run-time file with our own. On the Sun4 machines, we used the "-v" (verbose) flag with our link command:

```
g++ -v myApplication.o MSHN_syscall_lib.o -o myApp.
```

This generates the following output to the screen:

```
          /usr/local/lib/gcc-lib/sparc-sun-sunos4.1/2.6.3/ld -e start
-dc -dp -o myApp /lib/crt0.o -L/usr/local/lib/gcc-lib/sparc-sun-
sunos4.1/2.6.3 -L/usr/local/lib myApplication.o MSHN_syscall_lib.o -
lg++ -lgcc -lc -lgcc
```

We then replace the system's default link "/lib/crt0.o" **(or "/lib/crt1.o")** with "./MSHN_crt0.o" **(or "./MSHN_crt1.o")** and use this, as our second step, to generate our executable.

```
/usr/local/lib/gcc-lib/sparc-sun-sunos4.1/2.6.3/ld -e start -dc -dp -o
myApp ./MSHN_crt0.o -L/usr/local/lib/gcc-lib/sparc-sun-sunos4.1/2.6.3 -
L/usr/local/lib myApplication.o MSHN_syscall_lib.o -lg++ -lgcc -lc -
lgcc
```

or

```
/usr/local/lib/gcc-lib/sparc-sun-sunos4.1/2.6.3/ld -e start -dc -dp -o
myApp ./MSHN_crt1.o -L/usr/local/lib/gcc-lib/sparc-sun-sunos4.1/2.6.3 -
L/usr/local/lib myApplication.o MSHN_syscall_lib.o -lg++ -lgcc -lc -
lgcc
```

e. Compile and start the clock server. Enter the clock directory, compile clockServer.cc and then run clockServer. clockServer listens at a fixed port (#12391) defined in clockIncludes.h. Additionally, a variable is defined in clockIncludes.h if the server runs on a LINUX platform.

f. Run the application.

APPENDIX B. MSHN LIBRARY MAKEFILE2

Makefile for MSHN Client Library, modified to explicitly link with c++ and g++ compilers, the math and c libraries.

```
CC= g++

#uncomment for SUN6 CC=CC

MSHN_syscall_lib.o: MSHN_monitor_RRD_Class.o MSHN_utility.o
MSHN_MAINc.o\
	hashClass.o MSHN_MAIN.o MSHN_network_IO.o
MSHN_export_RSS_Class.o
	${CC} -O3 -c MSHN_syscall_lib.cc -o
MSHN_syscall_temp.o
#	ld -i -g MSHN_syscall_temp.o MSHN_monitor_RRD_Class.o
MSHN_utility.o \
#	hashClass.o MSHN_MAIN.o MSHN_network_IO.o
MSHN_export_RSS_Class.o \
#	-L./ -lMSHNc -o MSHN_syscall_lib.o
	ld -r -t -Bstatic MSHN_syscall_temp.o
MSHN_monitor_RRD_Class.o \
	MSHN_utility.o hashClass.o MSHN_MAIN.o
MSHN_network_IO.o MSHN_MAINc.o \
	MSHN_export_RSS_Class.o -L./ -lMSHNc -L /usr/local/lib
-lg++ -lstdc++ -lm -lc -o MSHN_syscall_lib.o
	rm MSHN_syscall_temp.o
	rm MSHN_utility.o
	rm MSHN_monitor_RRD_Class.o
	rm hashClass.o
	rm MSHN_MAIN.o
	rm MSHN_export_RSS_Class.o
	rm MSHN_network_IO.o
	rm MSHN_MAINc.o

MSHN_monitor_RRD_Class.o: MSHN_monitor_RRD_Class.cc
MSHN_utility.h \
	MSHN_types.h
	${CC} -O3 -c MSHN_monitor_RRD_Class.cc

MSHN_MAIN.o: MSHN_MAIN.cc
	${CC} -O3 -c MSHN_MAIN.cc -o MSHN_MAIN.o

MSHN_MAINc.o: MSHN_MAINc.c
	gcc -c MSHN_MAINc.c -o MSHN_MAINc.o
```

```
MSHN_network_IO.o:  MSHN_network_IO.cc
     ${CC} -O3 -c MSHN_network_IO.cc

MSHN_utility.o: MSHN_utility.cc MSHN_types.h
     ${CC} -O3 -c MSHN_utility.cc

hashClass.o: hashClass.cc
     ${CC} -O3 -c hashClass.cc

MSHN_export_RSS_Class.o: MSHN_export_RSS_Class.cc
     ${CC} -O3 -c MSHN_export_RSS_Class.cc
```

APPENDIX C. MSHN_SYSCALL_LIB.CC

```
//************************************************************
// File: MSHN_syscall_lib.cc
// Name: Matt Schnaidt (modified with permission by W. Porter
// Operating Environment: Linux 2.0.29, SUN OS
// Compiler: g++ for Linux, Unix
// Last Modified: 7 Feb 99
//
// Description:  When this file is included in another program, it
//   causes all calls from that program to read, write or exit, to
//   be "caught", information
//   recorded, and passed on to the operating system.
// Inputs:  The operating system calls.
// Outputs:  none.
// Process:  none
// Assumptions:  Calling progams input proper data type.
// Warnings:  no return function returns.  This warning cannot be
//   gotten rid of without major work-arounds.
//
//************************************************************
//
#include <stdio.h>
#include <iostream.h>
#include <sys/types.h>
#include <netdb.h>              //for gethostent in accept
#include <strings.h>
#include "MSHN_syscall_lib.h"
#include "MSHN_monitor_RRD_Class.h"
#include "MSHN_export_RSS_Class.h"
#include "MSHN_utility.h"
#include "MSHN_types.h"
#include "hashClass.h"
#include "clockIncludes.h"
#include "MSHN_network_IO.h"

//OBJECT declarations
//the object that tracks all resource usage
extern MSHN_monitor_RRD_Class resourceMonitor;
MSHN_export_RSS_Class rssObject;

//the object that keeps track of what the type is of each fd
hashClass fdTable;


//define prototypes for wrapper functions
void exitWrapper(int, void(*)(int));
int readWrapper(int, char*, int, int(*)(int, char*, int));
int writeWrapper(int, char*, int, int(*)(int, char*, int));
int closeWrapper(int, int(*)(int));

int openWrapper(const char*, int, int, int(*)(const char*, int, int));

int socketWrapper(int, int, int, int(*)(int, int, int));
int acceptWrapper(int, struct sockaddr*, int*,

                  int(*)(int, struct sockaddr*, int*));

int connectWrapper(int, struct sockaddr*, int,

                   int(*)(int, struct sockaddr*, int));


//declare the redefined clib symbols as available externally
extern "C"{
   //these must be defined for LINUX
   #ifdef LINUX
   extern int __READ(int, char*, int);
   extern int __WRITE(int, char*, int);
   extern int __OPEN(const char*, int, int);
   extern int __CLOSE(int);
   #endif


   //these must be defined for sun 5.5
   #ifdef SUN55
   extern int _READ(int, char*, int);
   extern int _WRITE(int, char*, int);
   extern int _OPEN(const char*, int,int);
   extern int _CLOSE(int);
   extern int _SOCKET(int, int, int);
   extern int _ACCEPT(int, struct sockaddr*, int*);
   extern int __ACCEPT(int, struct sockaddr*, int*);
   extern int _CONNECT(int, struct sockaddr*, int);
   extern int _CONNECT2(int, struct sockaddr*, int);
   #endif


   //these are required for Linux, Sun 4.x, Sun 5.x
   extern int READ(int, char*, int);
   extern int WRITE(int, char*, int);
   extern int OPEN(const char*, int,int);
   extern int CLOSE(int);
   extern void EXIT(int);
   extern void _EXIT(int);
   extern int SOCKET(int, int, int);
   extern int LISTEN(int, int);
   extern int ACCEPT(int, struct sockaddr*, int*);
   extern int CONNECT(int, struct sockaddr*, int);
   extern int SEND(int, const void*, int, unsigned int);
   extern int SENDTO(int, const void*, int, unsigned int,
                             const struct sockaddr*, int);
   extern int SENDMSG(int, const struct msghdr*, unsigned int);
   extern int RECV(int, void*, int, unsigned int);
   extern int RECVFROM(int, void*, int, unsigned int, struct sockaddr*,
int*);
   extern int RECVMSG(int, struct msghdr*, unsigned int);
}//end extern "C"
```

```
//==================================================================
=//

//===============NON-ARCHITECTURE SPECIFIC WRAPPER
FUNCTIONS============//

//==================================================================
=//
//

//--------------------------------------------------------

// Function: void exitWrapper()

// Purpose:  Outputs to the RRD this application's name and

//    resource utilization, then calls the system's exit

//    function.

//---------------------------------------------------------

//

void exitWrapper(int status, void(*exitFunction)(int))
{

   #ifdef MSHN_DEBUG
      printf("--inside exitWrapper--\n");
   #endif

   resourceMonitor.updateResourceServer(status);

   //pass the system call on
   (*exitFunction)(status);

   //will not return
}//end exitWrapper

//--------------------------------------------------------

// Function: readWrapper()

// Purpose: Calls the system's read system call, calls the

//          resource monitor which updates number of reads

//          and number of bytes read.

//---------------------------------------------------------
int readWrapper(int fd, char* buf, int len,
                int(*readFunction)(int, char*, int))
{

   //used to measure duration of reads
   struct timeval startTime,
                  endTime;
   int returnValue = 0,
       tempValue = 0;

   double readDuration;

   bucketElement* fdPtr = fdTable.lookUp(fd);
   fd_type thisFdType;

   //if ptr is null, then this is std io
   if (fdPtr){
      thisFdType = fdPtr->type;
   }
   else{
      thisFdType = TERMINAL_IO;
   }//end if

   #ifdef MSHN_DEBUG
     cout<<"inside readWrapper, fd => "<<fd<<", FdType=>
"<<thisFdType<<endl;
   #endif

   if(thisFdType==LOCAL_FILE){

       //start the clock, make the system call, and stop the clock
       returnValue = (*readFunction)(fd, buf, len);

       //update the read counters
       resourceMonitor.updateLocReads(returnValue);
   }
   //if this is a read from across the network, check latency and b/w
   else if(thisFdType == NETWORK_IPC){

      //start the clock, make the system call, and stop the clock
      startTime   = getClockTime();
      returnValue = networkRead(fd, buf, len, startTime,

                                          fdPtr, readFunction);
      endTime     = getClockTime();

      //update the terminal read/write information
      double elapsedTime =(calcTimeDiff(&startTime, &endTime));

      //update the network reads
      resourceMonitor.updateNetReads(returnValue, elapsedTime);
   }
   else if(thisFdType == LOCAL_IPC){
      returnValue = (*readFunction)(fd, buf, len);
      resourceMonitor.updateLocIPCReads(returnValue);
   }
   //if this is a file accessed over the network, collect access
   // time data
   else if(thisFdType == TERMINAL_IO){
      //start the clock, make the system call, and stop the clock
      startTime   = getClockTime();
      returnValue = (*readFunction)(fd, buf, len);
      endTime     = getClockTime();
      //calculate the time it took to do the read
      double elapsedTime =(calcTimeDiff(&startTime, &endTime));
      //update the read counters
      resourceMonitor.updateTerminalReads(returnValue, elapsedTime);
   }
   else if(thisFdType == REMOTE_FILE){
      //start the clock, make the system call, and stop the clock
      startTime   = getClockTime();
      returnValue = (*readFunction)(fd, buf, len);
      endTime     = getClockTime();

      //calculate the time it took to do the read
      double elapsedTime =(calcTimeDiff(&startTime, &endTime));
      resourceMonitor.updateDistReads(returnValue, elapsedTime);
   }
   else{
      returnValue = (*readFunction)(fd, buf, len);
      resourceMonitor.updateLocIPCReads(returnValue);
   }//end if

   #ifdef MSHN_DEBUG
     cout<<"--about to leave read wrapper,read # bytes "<<returnValue;
     cout<<" --"<<endl<<endl;
   #endif

   //returns the size of the read
   return(returnValue);
}//end readWrapper()



//-----------------------------------------------------------

// Function: writeWrapper()

// Purpose: Calls the system's write system call, calls the

//          resource monitor which updates number of writes

//          and number of bytes written.

//------------------------------------------------------------

//

int writeWrapper(int fd, char* buf, int len,

                 int(*writeFunction)(int, char*, int))

{

   int returnValue;
```

```
bucketElement* fdPtr = fdTable.lookUp(fd);
fd_type thisFdType;


//if ptr is null, then this is std io
if (fdPtr){
   thisFdType = fdPtr->type;
}
else{
   thisFdType = TERMINAL_IO;
}//end if


//if this is a write from across the network, check latency and b/w
if(thisFdType == NETWORK_IPC){


   //append flags to the front and rear of the message
   returnValue = networkWrite(fd, buf, len, writeFunction);


   resourceMonitor.updateNetWrites(returnValue);
}
else{
   //pass the system call on
   returnValue = (*writeFunction)(fd, buf, len);


   if(thisFdType==LOCAL_FILE){
      resourceMonitor.updateLocWrites(returnValue);
   }
   else if(thisFdType == TERMINAL_IO){
      resourceMonitor.updateTerminalWrites(returnValue);
      }
      else if(thisFdType == REMOTE_FILE){
         resourceMonitor.updateDistWrites(returnValue);
      }
      else{ //by elimination, must be local IPC
         resourceMonitor.updateLocIPCWrites(returnValue);
      }//end if
   }//end if

   //returns the size of the write
   return(returnValue);
}//end writeWrapper()



//-----------------------------------------------------------
// Function:  closeWrapper()
// Purpose:  Calls the system's close call, cleans up the
//    fd data structure.
//------------------------------------------------------------
//
int closeWrapper(int fd, int(*closeFunction)(int))
{
   int returnValue = (*closeFunction)(fd);

   #ifdef MSHN_DEBUG
     cout<<"--inside closeWrapper--"<<endl;
   #endif
   if (returnValue == 0){
      fdTable.remove(fd);
   }//end if

   #ifdef MSHN_DEBUG
      fdTable.printHashTable();
      cout<<"--about to leave close wrapper--"<<endl<<endl;
   #endif

   return(returnValue);

}//end closeWrapper()



//-----------------------------------------------------------
// Function:  openWrapper()
// Purpose:  Calls the system's open system call.  Evals what
//   type of file is opened (if the open is successful), and
//   stores this information in the fdTable hash table data
//   structure.
//------------------------------------------------------------
//
int openWrapper(const char *file_name, int flag, int mode,
                int(*openFunction)(const char*, int, int))
{
   int returnValue = (*openFunction)(file_name, flag, mode);

   #ifdef MSHN_DEBUG
```

```
      cout<<"--inside openWrapper: fd=> "<<returnValue<<"file=> ";
      cout<<file_name<< " --"<<endl;
    #endif

    if (returnValue > 0){
      fdTable.make_entry(returnValue, getFDType(returnValue));
    }//end if

    #ifdef MSHN_DEBUG
      fdTable.printHashTable();
      cout<<"--about to leave open wrapper--"<<endl<<endl;
    #endif

    return(returnValue);

}//end openWrapper()



//---------------------------------------------------------
// Function: socketWrapper()
// Purpose:  Calls the system's socket sys call. Evaluates what
//   type of socket is opened (if the open is successful), and
//   stores this information in the fdTable hash table data
//   structure.
//----------------------------------------------------------
//
int socketWrapper(int domain, int type, int protocol,
                  int(*socketFunction)(int, int, int))
```

```
{
   unsigned long args[3];
   args[0] = domain;
   args[1] = type;
   args[2] = protocol;


   int returnValue = (*socketFunction)(domain, type, protocol);

   fd_type fdType;

   #ifdef MSHN_DEBUG
     cout<<"--inside socketWrapper--"<<endl;
     cout<<"return value => "<<returnValue<<endl;
   #endif

   if (returnValue > 0){

     //need to refine this to catch bind/access syscall to ip addr
     if(domain == 2){                         //2==PF_INET

        fdType = NETWORK_IPC;
     }
     else{
        fdType = LOCAL_IPC;
     }//end if

     fdTable.make_entry(returnValue, fdType);
   }//end if

   #ifdef MSHN_DEBUG
      fdTable.printHashTable();
      cout<<"--about to leave socketWrapper--"<<endl<<endl;
   #endif

   return(returnValue);

}//end socketWrapper()


//----------------------------------------------------------
// Function: acceptWrapper()
// Purpose:  Calls the system's accept system call, Evaluates what
//    type of accept is opened (if the open is successful), and
//    stores this information in the fdTable hash table data
//    structure.
//-----------------------------------------------------------
//
int acceptWrapper(int socket, struct sockaddr* addr, int* addrlen,
                  int(*acceptFunction)(int, struct sockaddr*, int*))
{
   int returnValue;

   fd_type fdType;
   bucketElement* fdTablePtr;


   //pass on the system call

   returnValue = (*acceptFunction)(socket, addr, addrlen);


   #ifdef MSHN_DEBUG

     cout<<"--inside acceptWrapper--"<<endl;

     cout<<"return value => "<<returnValue<<endl;

   #endif


   if (returnValue > 0){


      unsigned long acceptAddress;

      int isLocal;


      //this is something of a hack; normally, would dereference the

      // struct sockaddr* addr, but we'd need to include socket.h, and

      // that would screw up our redefinition of the socket calls.
This

      // function gets the address of the machine which we accepted a

      // connection

      char* tempPtr = (char*)addr;

//memcpy(&acceptAddress, (tempPtr + sizeof(short) + sizeof(u_short)),
4);

      //isLocal = resourceMonitor.isHostLocal(acceptAddress);


      isLocal = 1; // ensures reads, writes are local
//update the appropriate entries in the fd table

      if(isLocal){

       //create an entry for this fd in the fd hash table; include
```

```
        // fd type
          fdTable.make_entry(returnValue, LOCAL_IPC);
      }
      else{
        //create an entry for this fd in the fd hash table; enter
        // fd type, ip address of remote machine, clockOffset between
        // the remote machine and this machine.
          fdTablePtr = fdTable.make_entry(returnValue, NETWORK_IPC);
          fdTablePtr->ipAddress   = acceptAddress;
          fdTablePtr->clockOffset = getClockOffset(acceptAddress, PORT,
3);
      }//end if

   }//end if

   #ifdef MSHN_DEBUG
      fdTable.printHashTable();
      cout<<"--about to leave acceptWrapper--"<<endl<<endl;
   #endif
   return(returnValue);

}//end acceptWrapper()


//----------------------------------------------------------
// Function: connectWrapper(int, struct sockaddr*, int*)
// Purpose:  Calls the system's connect system call, Evaluates what
//   type of connect is opened (if the open is successful), and
//   stores this information in the fdTable hash table data
//   structure.
//------------------------------------------------------------
//
int connectWrapper(int socket, struct sockaddr* addr, int addrlen,
                   int(*connectFunction)(int, struct sockaddr*, int))
{
   int returnValue;

   fd_type fdType;

   bucketElement* fdTablePtr;

   #ifdef MSHN_DEBUG
     cout<<"--inside connect wrapper--"<<endl;
   #endif

   //pass on the system call
   returnValue = (*connectFunction)(socket, addr, addrlen);

   if (returnValue == 0){

      long connectAddress;
      int isLocal;

      //this is something of a hack; normally, would dereference the
      // struct sockaddr* addr, but we'd need to include socket.h, and
      // that would screw up our redefinition of the socket calls.
This
      // function gets the address of the machine which we connected a
      // connection

      char* tempPtr = (char*)addr;

     // memcpy(&connectAddress, (tempPtr + sizeof(short) +
sizeof(u_short)), 4);

      //isLocal = resourceMonitor.isHostLocal(connectAddress);


      isLocal = 1; //ensures reads,writes are local
if(isLocal){

       //create an entry for this fd in the fd hash table; include

       // fd type

         fdTable.make_entry(socket, LOCAL_IPC);

      }

      else{

       //create an entry for this fd in the fd hash table; enter

       // fd type, ip address of remote machine, clockOffset between

       // the remote machine and this machine.

         fdTablePtr = fdTable.make_entry(socket, NETWORK_IPC);

         fdTablePtr->ipAddress   = connectAddress;

         fdTablePtr->clockOffset = getClockOffset(connectAddress, PORT,
5);

      }//end if


   }//end if


   #ifdef MSHN_DEBUG

      fdTable.printHashTable();

      cout<<"--about to leave _connect wrapper--"<<endl<<endl;

   #endif


   return(returnValue);
```

```
}//end connectWrapper()



//======================================================================
=//

//===============LINUX SPECIFIC WRAPPER-CALLING
FUNCTIONS===============//

//======================================================================
=//

//

#ifdef LINUX
//-----------------------------------------------------------
// Function: __read()
// Purpose:  "catches" the read system call, calls readWrapper()

//------------------------------------------------------------
//
int __read(int fd, char* buf, int len)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A __read()"<<endl;
   #endif
   return(readWrapper(fd, buf, len, __READ));

}//end __read()


//-----------------------------------------------------------
// Function: __write(int fd, char* buf, int len)
// Purpose:  "catches" the write system call, calls writeWrapper().
//------------------------------------------------------------
//
int __write(int fd, char* buf, int len)
{

   return(writeWrapper(fd, buf, len, __WRITE));

}//end __write()



//-----------------------------------------------------------
// Function:  __close(int fd)
// Purpose:  "catches" the __close system call. Calls

//   closeWrapper().
//------------------------------------------------------------
//
int __close(int fd){
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A __close()"<<endl;

   #endif

   return(closeWrapper(fd, __CLOSE));
}//end __close
#endif




//=====================================================================
=//

//===============SUN 5.5 SPECIFIC WRAPPER-CALLING
FUNCTIONS=============//

//=====================================================================
=//

//
#ifdef SUN55
//------------------------------------------------------------
// Function: _read(int fd, char* buf, int len)
// Purpose:  "catches" the _read system call, calls the
//   readWrapper().
//-------------------------------------------------------------
//
int _read(int fd, char* buf, int len)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A _read()"<<endl;
   #endif
   return(readWrapper(fd, buf, len, _READ));

}//end _read(int fd, char* buf, int len




//------------------------------------------------------------

// Function: _write(int fd, char* buf, int len)

// Purpose:  "catches" the _write system call, calls the

//   writeWrapper().

//-------------------------------------------------------------

//

int _write(int fd, char* buf, int len)
```

```
{

   return(writeWrapper(fd, buf, len, _WRITE));


}//end _write(int fd, char* buf, int len



//---------------------------------------------------------

// Function:  _open(const char *file_name, int flag, int mode)

// Purpose:  "catches" the _open system call, calls the

//   openWrapper().

//----------------------------------------------------------

//

int _open(const char *file_name, int flag, int mode)

{

   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A _open()"<<endl;

   #endif

   return(openWrapper(file_name, flag, mode, _OPEN));


}//end _open



//---------------------------------------------------------

// Function:  _close(int fd)

// Purpose:  "catches" the _close system call.  Calls the

//   closeWrapper().

//----------------------------------------------------------

//
```

```
int _close(int fd)

{

   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A _close()"<<endl;

   #endif

   return(closeWrapper(fd, _CLOSE));

}//end _close




//------------------------------------------------------------

// Function: _socket(int domain, int type, int protocol)

// Purpose:  "catches" the _socket system call, calls the

//    socketWrapper().

//-------------------------------------------------------------

//

int _socket(int domain, int type, int protocol)

{

   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A _socket()"<<endl;

   #endif

   return(socketWrapper(domain, type, protocol, _SOCKET));


}//end _socket(int domain, int type, int protocol)




//------------------------------------------------------------

// Function: _accept(int socket, struct sockaddr* addr, int* addrlen)

// Purpose:  "catches" the _accept system call, calls the
//   acceptWrapper().
//-----------------------------------------------------------
//
int _accept(int socket, struct sockaddr* addr, int* addrlen)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A _accept()"<<endl;
   #endif
   return(acceptWrapper(socket, addr, addrlen, _ACCEPT));

}//end _accept(int socket, struct sockaddr* addr, int* addrlen)



//-----------------------------------------------------------
// Function: __accept(int socket, struct sockaddr* addr, int* addrlen)
// Purpose:  "catches" the __accept system call, calls the
//   acceptWrapper().
//-----------------------------------------------------------
//
int __accept(int socket, struct sockaddr* addr, int* addrlen)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A __accept()"<<endl;
   #endif
   return(acceptWrapper(socket, addr, addrlen, __ACCEPT));

}//end __accept(int socket, struct sockaddr* addr, int* addrlen)
```

```
//--------------------------------------------------------------
// Function: _connect(int socket, struct sockaddr* addr, int* addrlen)
// Purpose:  "catches" the _connect system call, calls the
//   connectWrapper().
//---------------------------------------------------------------
//
int _connect(int socket, struct sockaddr* addr, int addrlen)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A _connect()"<<endl;
   #endif
   return(connectWrapper(socket, addr, addrlen, _CONNECT));

}//end _connect(int socket, struct sockaddr* addr, int* addrlen)


//--------------------------------------------------------------
// Function: _connect2(int socket, struct sockaddr* addr, int* addrlen)
// Purpose:  "catches" the _connect2 system call, calls the
//   connectWrapper().
//---------------------------------------------------------------
//
int _connect2(int socket, struct sockaddr* addr, int addrlen)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A _connect2()"<<endl;
   #endif
   return(connectWrapper(socket, addr, addrlen, _CONNECT2));
```

```
}//end _connect2(int socket, struct sockaddr* addr, int* addrlen)

#endif



//======================================================================
=//

//==========NON-ARCHITECTURE SPECIFIC WRAPPER-CALLING
FUNCTIONS=========//

//======================================================================
=//

//
//-------------------------------------------------------------
// Function: read(int fd, char* buf, int len)
// Purpose:  "catches" the read system call, calls the
//   readWrapper().

//--------------------------------------------------------------
//
int read(int fd, char* buf, int len)
{
   #ifdef MSHN_DEBUG
     cout<<"CAUGHT A read()"<<endl;
   #endif
   return(readWrapper(fd, buf, len, READ));
}//end read(int fd, char* buf, int len


//-------------------------------------------------------------
// Function: write(int fd, char* buf, int len)
// Purpose:  "catches" the write system call, calls the
//   writeWrapper().

//--------------------------------------------------------------
//
int write(int fd, char* buf, int len)
{
   return(writeWrapper(fd, buf, len, WRITE));



}//end write(int fd, char* buf, int len


//-------------------------------------------------------------
// Function:  open(const char *file_name, int flag, int mode)
// Purpose:  "catches" the open system call, calls the

//   openWrapper().
```

```
//------------------------------------------------------------
//
int open(const char *file_name, int flag, int mode)
{
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A open()"<<endl;

   #endif

   return(openWrapper(file_name, flag, mode, OPEN));


}//end open


//-----------------------------------------------------------
// Function:  close(int fd)
// Purpose:  "catches" the close system call, calls the

//   closeWrapper().
//------------------------------------------------------------
//
int close(int fd)
{
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A close()"<<endl;

   #endif

   return(closeWrapper(fd, CLOSE));

}//end close


//-----------------------------------------------------------
// Function: socket(int domain, int type, int protocol)
// Purpose:  "catches" the socket system call, calls the

//   socketWrapper().
//------------------------------------------------------------
//
int socket(int domain, int type, int protocol)
{
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A socket()"<<endl;

   #endif

   return(socketWrapper(domain, type, protocol, SOCKET));


}//end socket(int domain, int type, int protocol)
```

```
//---------------------------------------------------------
// Function: accept(int socket, struct sockaddr* addr, int* addrlen)
// Purpose:  "catches" the accept system call, calls the

//   acceptWrapper().
//----------------------------------------------------------
//
int accept(int socket, struct sockaddr* addr, int* addrlen)
{
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A accept()"<<endl;

   #endif

   return(acceptWrapper(socket, addr, addrlen, ACCEPT));


}//end accept(int socket, struct sockaddr* addr, int* addrlen)


//---------------------------------------------------------
// Function: connect(int socket, struct sockaddr* addr, int* addrlen)
// Purpose:  "catches" the connect system call, calls the

//   connectWrapper
//----------------------------------------------------------
//
int connect(int socket, struct sockaddr* addr, int addrlen)
{
   #ifdef MSHN_DEBUG

     cout<<"CAUGHT A connect()"<<endl;

   #endif

   return(connectWrapper(socket, addr, addrlen, CONNECT));



}//end connect(int socket, struct sockaddr* addr, int* addrlen)


//---------------------------------------------------------
// Function: void _exit(int status)
// Purpose:  "catches" the exit system call, calls the

//   exitWrapper().
//----------------------------------------------------------
//
void _exit(int status)
{
   #ifdef MSHN_DEBUG
      printf("--inside _exit wrapper--\n");
   #endif
   //pass the system call on
   exitWrapper(status, _EXIT);
   //will not return
}//end _exit(int status)


//----------------------------------------------------------
// Function: void exit()
// Purpose:  "catches" the exit system call, calls the

//   exitWrapper() function.
//-----------------------------------------------------------
//
void exit(int status)
{
   #ifdef MSHN_DEBUG
      printf("--inside exit function--\n");
   #endif
   exitWrapper(status, EXIT);


}//end exit(int status)


//end file MSHN_syscall_lib.cc
```

APPENDIX D. MSHN_MONITOR_RRD_CLASS.CC

Code for sending results of MSHN Client Library Wrapper to an files.

```
//*************************************************************
// File: MSHN_monitor_RRD_Class.cc
// Name: Matt Schnaidt (modified with permission by W Porter)
// Operating Environment: Linux 2.0.29, Unix 4.1.4
// Compiler: g++ for Linux, Sun OS 4.1.4
// Date: 25 Feb 99
//
// Description:  Records resource usage information, and
//   updates the resource server when called.
// Inputs:  Number of bytes read or written.
// Assumptions:  Calling progams input proper data type.
// Warnings:  none.
//*************************************************************
//
#include <stdio.h>              //for FILE
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/resource.h>       //for rusage
#include <netdb.h>              //for get hostent
#include <unistd.h>             //for gethostname
#include <netinet/in.h>         //for htonl()
#include "MSHN_monitor_RRD_Class.h"
#include "MSHN_types.h"
#include "MSHN_utility.h"

//local prototype
char* getPsData(int, char*);

#ifdef SUN55

extern "C" {
   extern int getrusage(int, struct rusage*);
}//end extern "C"
#endif


#ifdef SUN4
extern "C" {
   extern int getrusage(int, struct rusage*);
}//end extern "C"
#endif


//--------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::MSHN_monitor_RRD_Class():
// Return Value:  none.
// Parameter:  none.
// Purpose:  User-defined default constructor; initializes
//   resource data members.
```

```
//------------------------------------------------------------
//
MSHN_monitor_RRD_Class::MSHN_monitor_RRD_Class():
   sizeLocReads(0), numLocReads(0),
   sizeLocWrites(0), numLocWrites(0), sizeDistReads(0),
numDistReads(0),
   sizeDistWrites(0), numDistWrites(0), timeDistReads(0.0),
   numTerminalReads(0), sizeTerminalReads(0), timeTerminalReads(0.0),
   numTerminalWrites(0), sizeTerminalWrites(0), numNetReads(0),
   sizeNetReads(0), sizeNetWrites(0), numNetWrites(0),
timeNetReads(0.0),
   thisAddress(0)
{

   const int HOST_NAME_SIZE = 100;
   char hostName[HOST_NAME_SIZE];
   gethostname(hostName, HOST_NAME_SIZE);
   struct hostent* thisHostPtr = gethostbyname(hostName);
   memcpy(&thisAddress, thisHostPtr->h_addr, thisHostPtr->h_length);
}//end MSHN_monitor_RRD_Class::MSHN_monitor_RRD_Class()


//destructor
MSHN_monitor_RRD_Class::~MSHN_monitor_RRD_Class(){

}//end MSHN_monitor_RRD_Class::~MSHN_monitor_RRD_Class()

//-----------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::isHostLocal(const int& HOST)
// Return Value:  int, 1 if host is local, 0 otherwise.
// Parameter:  unsigned long, HOST, the host address to compare
//             to this one. MUST be in network byte ordering.
// Purpose:  If HOST is the same address as this machine,
//   returns 1; else returns 0.
//------------------------------------------------------------
//
int MSHN_monitor_RRD_Class::isHostLocal(const unsigned long& HOST)
{

 return(thisAddress == HOST);

}//end int MSHN_monitor_RRD_Class::isHostLocal(const int& HOST)


//-----------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateLocReads(int& len)
// Return Value:  none.
// Parameter:  int, the size of the read.
// Purpose:  Updates total bytes read locally and total number of
//   reads.
//------------------------------------------------------------
//
void  MSHN_monitor_RRD_Class::updateLocReads(int& len)
{
   //only increase if read is successful
   if (len>0){
      sizeLocReads += len;
   }//end if

   //count number of reads -- successful or not
   numLocReads++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateLocRead--"<<endl;
      cout<<"Total size of local reads = " << sizeLocReads << endl;
      cout<<"--leaving updateLocRead--"<<endl<<endl;
   #endif

  return;
}//end MSHN_monitor_RRD_Class::updateLocReads(int&)


//---------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::updateLocWrites(int& len)
// Return Value:  none.
// Parameter:  int, the size of the write.
// Purpose:  Updates total bytes written locally and total number of
//   writes.
//---------------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateLocWrites(int& len)
{
   //only increase if write is successful
   if(len>0){
      sizeLocWrites += len;
   }//end if

   //count number of reads -- successful or not
   numLocWrites++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateLocWrites--"<<endl;
      cout<<"Size of local writes = " << sizeLocWrites << endl;
      cout<<"--leaving updateLocWrites--"<<endl<<endl;
   #endif


}//end MSHN_monitor_RRD_Class::updateLocWrites(int&)


//---------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateDistReads(int& len, double&
time)
// Return Value:  none.
// Parameter:  int, the size of the read.
// Purpose:  Updates total bytes read locally and total number of
//   reads on network file server.
//---------------------------------------------------------------
//
void  MSHN_monitor_RRD_Class::updateDistReads(int& len, double& time)
{
   //only increase if read is successful
   if (len>0){
      sizeDistReads += len;
      timeDistReads += time;
   }//end if

   //count number of reads -- successful or not
   numDistReads++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateDistRead--"<<endl;
      cout<<"Total size of network fs reads = " << sizeDistReads <<
endl;
      cout<<"--leaving updateDistRead--"<<endl<<endl;
   #endif

  return;
}//end MSHN_monitor_RRD_Class::updateDistReads(int&, double& time)


//-------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::updateDistWrites(int& len)
// Return Value:  none.
// Parameter:  int, the size of the write.
// Purpose:  Updates total bytes written  and total number of
//   writes on network file server.
//--------------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateDistWrites(int& len)
{
   //only increase if write is successful
   if(len>0){
      sizeDistWrites += len;
   }//end if

   //count number of reads -- successful or not
   numDistWrites++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateDistWrites--"<<endl;
      cout<<"Size of network fs writes = " << sizeDistWrites << endl;
      cout<<"--leaving updateDistWrites--"<<endl<<endl;
   #endif

}//end MSHN_monitor_RRD_Class::updateDistWrites(int&)


//-------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateNetReads(int& len, double&
time)
// Return Value:  none.
// Parameter:  int, the size of the read.
// Purpose:  Updates total bytes read locally and total number of
//   reads on network file server.
//--------------------------------------------------------------
//
void  MSHN_monitor_RRD_Class::updateNetReads(int& len, double& time)
{
   //only increase if read is successful
   if (len>0){
      sizeNetReads += len;
      timeNetReads += time;
   }//end if

   //count number of reads -- successful or not
   numNetReads++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateNetRead--"<<endl;
      cout<<"Total size of network reads = " << sizeNetReads << endl;
      cout<<"--leaving updateNetRead--"<<endl<<endl;
   #endif

  return;
}//end MSHN_monitor_RRD_Class::updateNetReads(int&, double& time)


//---------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::updateNetWrites(int& len)
// Return Value:  none.
// Parameter:  int, the size of the write.
// Purpose:  Updates total bytes written  and total number of
//   writes on network file server.
//----------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateNetWrites(int& len)
{
   //only increase if write is successful
   if(len>0){
      sizeNetWrites += len;
   }//end if

   //count number of reads -- successful or not
   numNetWrites++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateNetWrites--"<<endl;
      cout<<"Size of network fs writes = " << sizeNetWrites << endl;
      cout<<"--leaving updateNetWrites--"<<endl<<endl;
   #endif

}//end MSHN_monitor_RRD_Class::updateNetWrites(int&)


//---------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateLocIPCReads(int& len)
// Return Value:  none.
// Parameter:  int, the size of the read.
// Purpose:  Updates total bytes read locally and total number of
//   reads on network file server.
//----------------------------------------------------------
//
void  MSHN_monitor_RRD_Class::updateLocIPCReads(int& len)
{
   //only increase if read is successful
   if (len>0){
      sizeLocIPCReads += len;
   }//end if

   //count number of reads -- successful or not
   numLocIPCReads++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateLocIPCRead--"<<endl;
      cout<<"Total size of network reads = " << sizeLocIPCReads <<
endl;
      cout<<"--leaving updateLocIPCRead--"<<endl<<endl;
   #endif

  return;
}//end MSHN_monitor_RRD_Class::updateLocIPCReads(int&)


//----------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::updateLocIPCWrites(int& len)
// Return Value:  none.
// Parameter:  int, the size of the write.
// Purpose:  Updates total bytes written  and total number of
//   writes on network file server.
//----------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateLocIPCWrites(int& len)
{
   //only increase if write is successful
   if(len>0){
      sizeLocIPCWrites += len;
   }//end if

   //count number of reads -- successful or not
   numLocIPCWrites++;

   if(0){
      cout<<endl<<"--inside updateLocIPCWrites--"<<endl;
      cout<<"Size of network fs writes = " << sizeLocIPCWrites << endl;
      cout<<"--leaving updateLocIPCWrites--"<<endl<<endl;
   }//end if

}//end MSHN_monitor_RRD_Class::updateLocIPCWrites(int&)


//----------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateTerminalReads(int& len,
double& time)
// Return Value:  none.
// Parameter:  int, the size of the read.
// Purpose:  Updates total bytes read and total number of
//   reads from terminal and time to do reads.
//----------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateTerminalReads(int& len, double&
time)
{
   //only increase if read is successful
   if (len>0){
      sizeTerminalReads += len;
      timeTerminalReads += time;
   }//end if

   //count number of reads -- successful or not
   numTerminalReads++;

   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateTerminalRead--"<<endl;
      cout<<"Total size of terminal reads = " << sizeTerminalReads <<
endl;
      cout<<"--leaving updateTerminalRead--"<<endl<<endl;
   #endif

  return;
}//end MSHN_monitor_RRD_Class::updateTerminalReads(int&, double& time)


//-------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class:: updateTerminalWrites(int& len)
// Return Value:  none.
// Parameter:  int, the size of the write.
// Purpose:  Updates total bytes write and total number of
//   writes from terminal and time to do writes.
//--------------------------------------------------------------
//
void MSHN_monitor_RRD_Class::updateTerminalWrites(int& len)
{
   //only increase if write is successful
   if (len>0){
      sizeTerminalWrites += len;
   }//end if

   //count number of writes -- successful or not
   numTerminalWrites++;

   if(0){
      cout<<endl<<"--inside updateTerminalWrite--"<<endl;
      cout<<"Total size of terminal writes = " << sizeTerminalWrites <<
endl;
      cout<<"--leaving updateTerminalWrite--"<<endl<<endl;
   }//end if

  return;
}//end MSHN_monitor_RRD_Class::updateTerminalWrites(int&)


//-------------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::updateResourceServer(int status)
// Return Value:  none.
// Parameter:  none.
// Purpose:  Updates the resource server with the total
//   resources used by this application.  Updates by
//   application name.  Currently, this is simply printed
//   to the screen.  Another student is developing a CORBA
//   persistent object to act as the resource server.
//--------------------------------------------------------------
```

```
//
void MSHN_monitor_RRD_Class::updateResourceServer(int status)
{
   int checkPid;               //this process' Pid

   //creates output file
   ofstream outputLoc("MSHN_outfile.txt" , ios::out);

   //if debug is set, output debug info
   #ifdef MSHN_DEBUG
      cout<<endl<<"--inside updateResourceServer--"<<endl;
   #endif

   #ifndef OUTPUT_TO_FILE
      ostream& outputLoc = cout;
   #endif

   appEndTime = getClockTime();

   outputLoc<<"Simulating update to Resource Requirements
Database"<<endl;
   outputLoc<<"Application name: "<<exeName<<endl;
   outputLoc<<"Input args: "<<inputArgs<<endl;

   if(status == 0){
      outputLoc<<"Application terminated normally"<<endl;
   }
   else{
      outputLoc<<"Application terminated abnormally, exit status =
"<<status<<endl;
   }//end if

   outputLoc<<"--------  LOCAL FILE DATA  ---------"<<endl;
   outputLoc<<"Total Bytes Read:    "<<sizeLocReads;
   outputLoc<<"  Written: "<<sizeLocWrites<<endl;
   outputLoc<<"Number of Reads:     "<<numLocReads;
   outputLoc<<"  Writes : "<<numLocWrites<<endl;

   outputLoc<<"--------  NETWORK FILE DATA  ---------"<<endl;
   outputLoc<<"Total Bytes Read:    "<<sizeDistReads;
   outputLoc<<"  Written: "          <<sizeDistWrites<<endl;
   outputLoc<<"Number of Reads:     "<<numDistReads;
   outputLoc<<"  Writes : "          <<numDistWrites<<endl;
   outputLoc<<"Total Time Reading:  "<<timeDistReads<<endl;

   outputLoc<<"--------  TERMINAL I/O DATA  ---------"<<endl;
   outputLoc<<"Total Bytes Read:    "<<sizeTerminalReads;
   outputLoc<<"  Written: "          <<sizeTerminalWrites<<endl;
   outputLoc<<"Number of Reads:     "<<numTerminalReads;
   outputLoc<<"  Writes : "          <<numTerminalWrites<<endl;
   outputLoc<<"Total Time Reading:  "<<timeTerminalReads<<endl;

   outputLoc<<"--------     NETWORK DATA    ---------"<<endl;
   outputLoc<<"Total Bytes Read:    "<<sizeNetReads;
   outputLoc<<"  Written: "          <<sizeNetWrites<<endl;
   outputLoc<<"Number of Reads:     "<<numNetReads;
   outputLoc<<"  Writes : "          <<numNetWrites<<endl;
   outputLoc<<"-------- LOCAL IPC  DATA    ---------"<<endl;
   outputLoc<<"Total Bytes Read:    "<<sizeLocIPCReads;
   outputLoc<<"  Written: "          <<sizeLocIPCWrites<<endl;
   outputLoc<<"Number of Reads:     "<<numLocIPCReads;
   outputLoc<<"  Writes: "           <<numLocIPCWrites<<endl;

   outputLoc<<endl<<"Application's Wall Clock Runtime:..";
   outputLoc<<calcTimeDiff(&appStartTime, &appEndTime);
   outputLoc<<" seconds"<<endl;

   //this determines and outputs how long the process ran
   struct rusage rusageStruct,
                      *rusagePtr = &rusageStruct;

   getrusage(RUSAGE_SELF, rusagePtr);
   double userTime = ((double)(rusagePtr->ru_utime.tv_usec)/1000000.) +
                            rusagePtr->ru_utime.tv_sec;
   double sysTime  = ((double)(rusagePtr->ru_stime.tv_usec)/1000000.) +
                            rusagePtr->ru_stime.tv_sec;
   outputLoc<<endl;
   outputLoc<<"system cpu time   = "<<sysTime<<" seconds"<<endl;
   outputLoc<<"user   cpu time   = "<<userTime<<" seconds"<<endl;
   outputLoc<<"max res set size  = "<<rusagePtr->ru_maxrss<<"
pages"<<endl;
   outputLoc<<"unshared memory   = "<<rusagePtr->ru_idrss<<"
pages"<<endl;
   outputLoc<<"page faults       = "<<rusagePtr->ru_majflt<<endl;
   outputLoc<<"---at program termination, the memory stats were:
"<<endl;

#ifdef SUN55
   int thisPid = getpid();
   outputLoc<<"num of physical pages in memory  = "<<getPsData(thisPid,
" osz")<<endl;
   outputLoc<<"num of pages in virtual memory   = "<<getPsData(thisPid,
"vsz")<<endl;
   outputLoc<<"num of pages in resident set     = "<<getPsData(thisPid,
"rss")<<endl;
#endif

   return;
}//end MSHN_monitor_RRD_Class:: updateResourceServer()


//---------------------------------------------------------
// Function:  MSHN_monitor_RRD_Class::getPsData
//                                   (int thisPid, char* psCommand)
// Return Value: char*
// Parameter:  thisPid, psCommand
// Purpose: gets process status (ps) data
//----------------------------------------------------------
//
char* getPsData(int thisPid, char* psCommand)
{

   char thisPidPtr[12];  //declares a char array for thisPid
   sprintf(thisPidPtr,"%d\0",thisPid); //places thisPid in char array

   char commandString[100] = "ps -p ";  //declares variable for ps
commands

   char *buffer = (char*) malloc(100*sizeof(char));  //declares string
for
                                                       //-buffer output

   strcat(commandString, thisPidPtr);
   strcat(commandString, " -o ");
   strcat(commandString, psCommand);

   char tempfilename[100] = "mshnTempFile";  //declares string for file
name
   strcat(tempfilename, thisPidPtr);
   strcat(commandString, "= > ");
   strcat(commandString, tempfilename);

   system(commandString);  //uses command string for system -ps call

   ifstream inFile(tempfilename);
   inFile>>buffer;

   char remove[100] = "rm";
   strcat(remove," ");
   strcat(remove, tempfilename);

   system(remove);  //removes file created to hold buffer contents
   return (buffer);

}//end getPsData()


//-----------------------------------------------------------
// Function:  MSHN_monitor_RRD_Class::enterExeName
//                                    (const char* thisName)
// Return Value:  none.
// Parameter:  thisName.
// Purpose: Sets exeName equal to thisName
//------------------------------------------------------------
//
   //assign the executable name to exeName
void MSHN_monitor_RRD_Class::enterExeName(const char* thisName)
{
  strcpy(exeName, thisName);
  return;
}//end MSHN_monitor_RRD_Class::enterExeName(const char* thisName)


//-----------------------------------------------------------
// Function:  MSHN_monitor_RRD_Class::getExeName
//                                    ()
// Return Value:  none.
// Parameter:  thisName.
// Purpose: gets thisName
//------------------------------------------------------------
```

```
//
char* MSHN_monitor_RRD_Class::getExeName()
{
   return (exeName);
}//end MSHN_monitor_RRD_Class::getExeName()


//---------------------------------------------------------
// Function:  MSHN_monitor_RRD_Class::enterInputArgs
//                                  (const char* theseArgs)
// Return Value:  none.
// Parameter:  char*, theseArgs a string containing all input
//   arguments.
// Purpose: Sets inputArgs equal to theseArgs
//----------------------------------------------------------
//
void MSHN_monitor_RRD_Class::enterInputArgs(const char* theseArgs)
{
   strcpy(inputArgs, theseArgs);
   return;
}//end MSHN_monitor_RRD_Class::enterInputArgs(const char* theseArgs)

//end file MSHN_monitor_RRD_Class.cc

//---------------------------------------------------------
// Function: MSHN_monitor_RRD_Class::enterAppStartTime
//                           (const struct timeval* startTime)
// Return Value:  none.
// Parameter: const struct timeval*, the start timeof the
//   application.
// Purpose: sets appStartTime equal to startTime.
//----------------------------------------------------------
//   //enter the start time of this application
void MSHN_monitor_RRD_Class::enterAppStartTime(const struct timeval*
startTime)
{
   appStartTime.tv_sec  = startTime->tv_sec;
   appStartTime.tv_usec = startTime->tv_usec;

}//end MSHN_monitor_RRD_Class::enterAppStartTime()
```

APPENDIX E. RUN SCRIPT FOR DEMO300

"sh" run script for Demo300 that launches EADSIM executables, sets clockserver for wrapper, and resets random number generator seed by tying it to the system clock.

```
#!/bin/sh

C3IDATA=/users/work4/nwporter/eadsim/data
export C3IDATA

Scenario=Demo300
if [ $# -eq 1 ]; then
  Scenario=$1
fi

CrashPath="/users/work4/nwporter/crash"
ExePath="/users/work4/nwporter/eadsim/execute/SUN2"

clockServer &

cd $CrashPath/c3i
$ExePath/c3i $Scenario 0 `date '+%H%M%S'` &

sleep 1

cd $CrashPath/det
$ExePath/detect $Scenario 0 `date '+%H%M%S'` &

sleep 16

cd $CrashPath/fp
$ExePath/fp $Scenario &
```

APPENDIX F. RUN SCRIPT FOR TRANSFER OF DATA

"sh" run script to transfer wrapper and simulation results for each Monte Carlo run to a separate directory. Script assigns unique "run number" extension to each file of output data associated with a given run.

```
#!/bin/sh

if [ ! "$#" -eq 1 ]
then
   echo "Usage: $0 version_number"
   exit 1
fi

RUN_VER="$1"
ROOT_DIR=/users/work4/nwporter/

cd ${ROOT_DIR:?}/eadsim
if [ -f mcreports/Threatsum.run${RUN_VER:?} ]
then
   echo "Version ${RUN_VER:?} of data files exist"
   exit 1
fi

cp Threatsum.engrpt mcreports/Threatsum.run${RUN_VER:?}
cp BlueAction.engrpt mcreports/BlueAction.run${RUN_VER:?}
cp RedAction.engrpt mcreports/RedAction.run${RUN_VER:?}
cd /users/work4/nwporter/crash/
cd c3i
cp MSHN_outfile.txt
/users/work4/nwporter/eadsim/mcreports/c3i_outfile.run${RUN
_VER:?}
cd /users/work4/nwporter/crash/
cd fp
cp MSHN_outfile.txt
/users/work4/nwporter/eadsim/mcreports/fp_outfile.run${RUN_
VER:?}
cd /users/work4/nwporter/crash/
cd det
cp MSHN_outfile.txt
/users/work4/nwporter/eadsim/mcreports/det_outfile.run${RUN
_VER:?}
cd /users/work4/nwporter/eadsim/run
cd c3i
```

```
cp Demo300_1.c3ipstat
/users/work4/nwporter/eadsim/mcreports/c3ipstat.run${RUN_VE
R:?}
cd /users/work4/nwporter/eadsim/run
cd detect
cp Demo300_1.detpstat
/users/work4/nwporter/eadsim/mcreports/detpstat.run${RUN_VE
R:?}
cd /users/work4/nwporter/eadsim/run
cd fp
cp Demo300_1.fppstat
/users/work4/nwporter/eadsim/mcreports/fppstat.run${RUN_VER
:?}
cp Demo300_1.stathdr
/users/work4/nwporter/eadsim/mcreports/stathdr.run${RUN_VER
:?}
```

APPENDIX G. RUN SCRIPT FOR TRANSFER OF DETERMINISTIC DATA

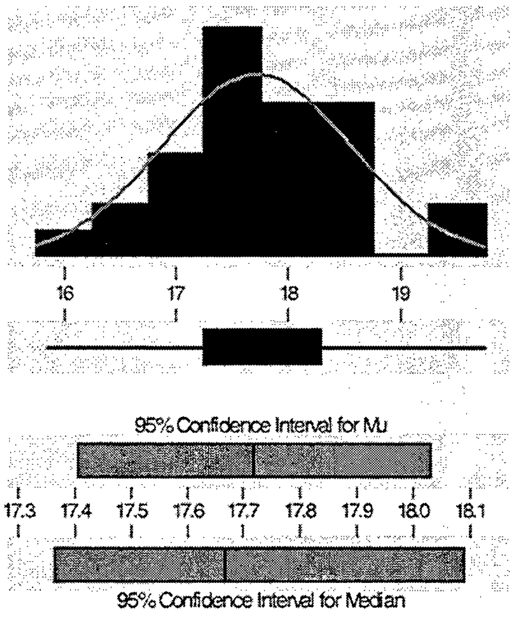
"sh" run script to transfer wrapper and simulation results for each deterministic run to a separate directory. Script assigns unique "run number" extension to each file of output data associated with a given run.

```
#!/bin/sh

if [ ! "$#" -eq 1 ]
then
   echo "Usage: $0 version_number"
   exit 1
fi

RUN_VER="$1"
ROOT_DIR=/users/work4/nwporter/

cd ${ROOT_DIR:?}/eadsim
if [ -f dreports/c3ipstat.run${RUN_VER:?} ]
then
   echo "Version ${RUN_VER:?} of data files exist"
   exit 1
fi

cd /users/work4/nwporter/eadsim/run
cd c3i
cp Demo300_1.c3ipstat
/users/work4/nwporter/eadsim/reports/c3ipstat.run${RUN_VER:
?}
cd /users/work4/nwporter/eadsim/run
cd detect
cp Demo300_1.detpstat
/users/work4/nwporter/eadsim/dreports/detpstat.run${RUN_VER
:?}
cd /users/work4/nwporter/eadsim/run
cd fp
cp Demo300_1.fppstat
/users/work4/nwporter/eadsim/dreports/fppstat.run${RUN_VER:
?}
```

APPENDIX H. MSHN WRAPPER OUTPUT: MONTE CARLO TRIALS

Note: Only data that generated a distribution is displayed.

Descriptive Statistics



Variable: C3I User CPU Time

Anderson-Darling Normality Test

A-Squared:	0.193
P-Value:	0.886
Mean	17.7177
StDev	0.8383
Variance	0.702667
Skewness	0.176778
Kurtosis	5.54E-03
N	30
Minimum	15.8600
1st Quartile	17.2450
Median	17.6650
3rd Quartile	18.2850
Maximum	19.7400

95% Confidence Interval for Mu

17.4047 18.0307

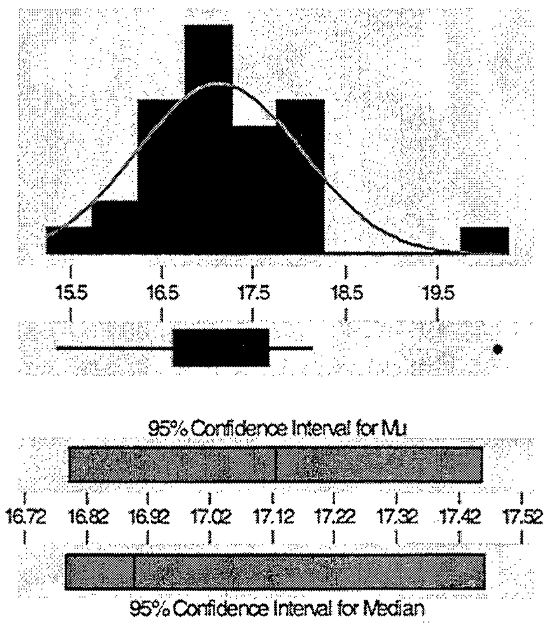
95% Confidence Interval for Sigma

0.6676 1.1269

95% Confidence Interval for Median

17.3637 18.0871

Descriptive Statistics



Variable: FP User CPU Time

Anderson-Darling Normality Test

A-Squared:	0.545
P-Value:	0.148
Mean	17.1253
StDev	0.8911
Variance	0.794012
Skewness	0.991135
Kurtosis	2.26742
N	30
Minimum	15.3600
1st Quartile	16.6200
Median	16.8950
3rd Quartile	17.6450
Maximum	20.1300

95% Confidence Interval for Mu

16.7926 17.4581

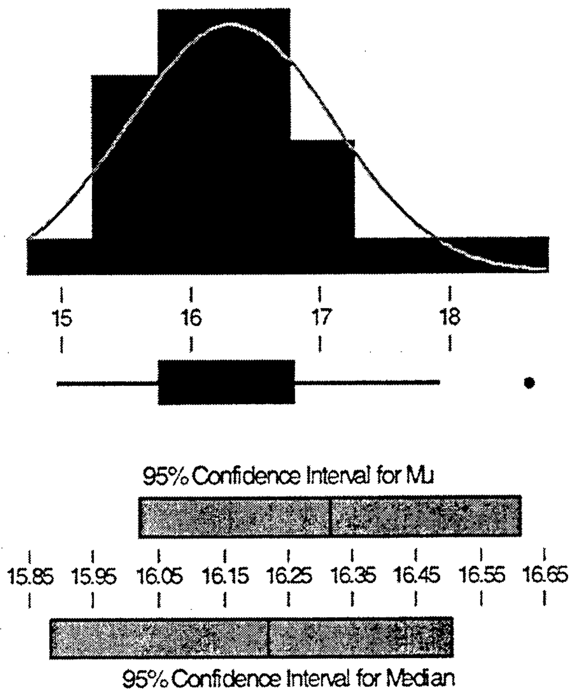
95% Confidence Interval for Sigma

0.7097 1.1979

95% Confidence Interval for Median

16.7869 17.4603

Descriptive Statistics



Variable: Detect User CPU Time

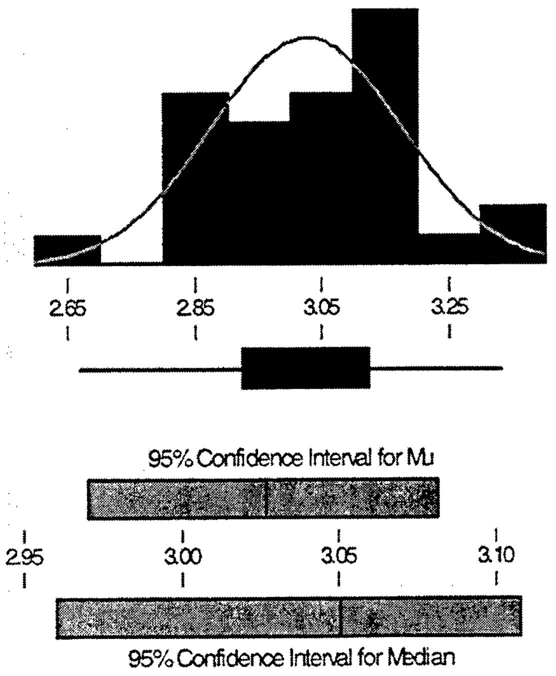
Anderson-Darling Normality Test	
A-Squared:	0.506
P-Value:	0.186
Mean	16.3160
StDev	0.7930
Variance	0.628887
Skewness	0.893720
Kurtosis	0.675003
N	30
Minimum	14.9700
1st Quartile	15.7625
Median	16.2150
3rd Quartile	16.7700
Maximum	18.6100

95% Confidence Interval for Mu	
16.0199	16.6121

95% Confidence Interval for Sigma	
0.6316	1.0661

95% Confidence Interval for Median	
15.8823	16.5026

Descriptive Statistics



Variable: C3I System CPU Time

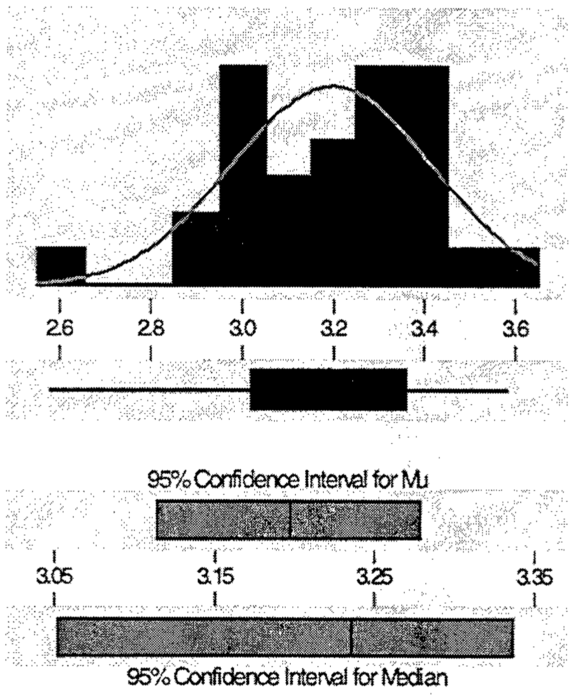
Anderson-Darling Normality Test	
A-Squared:	0.357
P-Value:	0.433
Mean	3.02600
StDev	0.14940
Variance	2.23E-02
Skewness	-2.0E-01
Kurtosis	-3.3E-01
N	30
Minimum	2.67000
1st Quartile	2.92250
Median	3.05000
3rd Quartile	3.12250
Maximum	3.33000

95% Confidence Interval for Mu	
2.97021	3.08179

95% Confidence Interval for Sigma	
0.11899	0.20085

95% Confidence Interval for Median	
2.96000	3.10771

Descriptive Statistics



Variable: FP System CPU Time

Anderson-Darling Normality Test

A-Squared:	0.513
P-Value:	0.179
Mean	3.19667
StDev	0.21979
Variance	4.83E-02
Skewness	-5.7E-01
Kurtosis	0.106161
N	30
Minimum	2.58000
1st Quartile	3.02000
Median	3.23500
3rd Quartile	3.35750
Maximum	3.58000

95% Confidence Interval for Mu

3.11460 3.27874

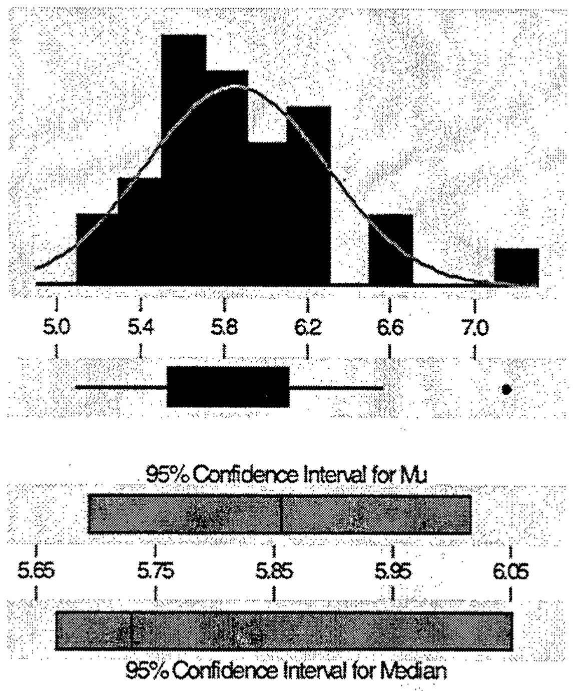
95% Confidence Interval for Sigma

0.17504 0.29546

95% Confidence Interval for Median

3.05229 3.33543

Descriptive Statistics



Variable: Detect System CPU Time

Anderson-Darling Normality Test

A-Squared:	0.528
P-Value:	0.164
Mean	5.85567
StDev	0.43027
Variance	0.185129
Skewness	0.833605
Kurtosis	0.858158
N	30
Minimum	5.10000
1st Quartile	5.53500
Median	5.73000
3rd Quartile	6.10250
Maximum	7.15000

95% Confidence Interval for Mu

5.69500 6.01633

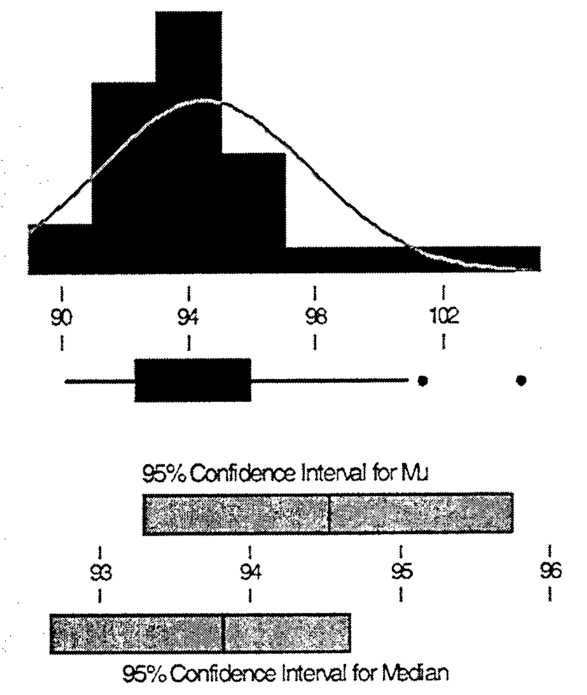
95% Confidence Interval for Sigma

0.34267 0.57841

95% Confidence Interval for Median

5.66686 6.05000

Descriptive Statistics



Variable: C3I Wall Clock Time

Anderson-Darling Normality Test

A-Squared:	1.158
P-Value:	0.004
Mean	94.5186
StDev	3.2978
Variance	10.8752
Skewness	1.23957
Kurtosis	1.19005
N	30
Minimum	90.150
1st Quartile	92.318
Median	93.814
3rd Quartile	95.862
Maximum	104.418

95% Confidence Interval for Mu

93.287 95.750

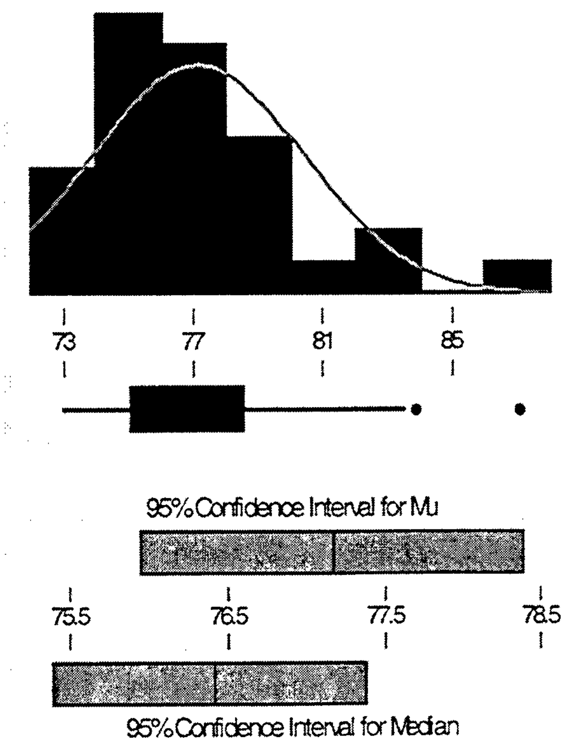
95% Confidence Interval for Sigma

2.626 4.433

95% Confidence Interval for Median

92.670 94.652

Descriptive Statistics



Variable: FP Wall Clock Time

Anderson-Darling Normality Test

A-Squared:	1.159
P-Value:	0.004
Mean	77.1691
StDev	3.2677
Variance	10.6780
Skewness	1.25967
Kurtosis	1.24295
N	30
Minimum	72.9853
1st Quartile	75.0810
Median	76.4142
3rd Quartile	78.5339
Maximum	87.0575

95% Confidence Interval for Mu

75.9489 78.3893

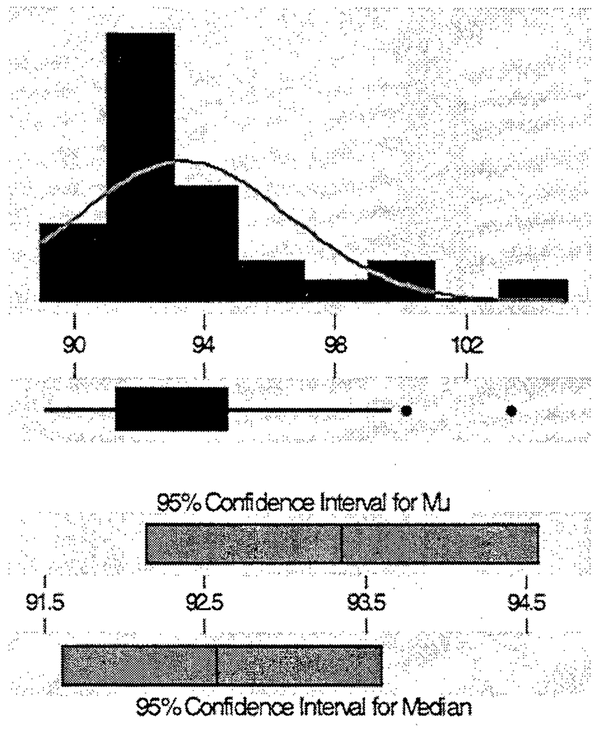
95% Confidence Interval for Sigma

2.6024 4.3928

95% Confidence Interval for Median

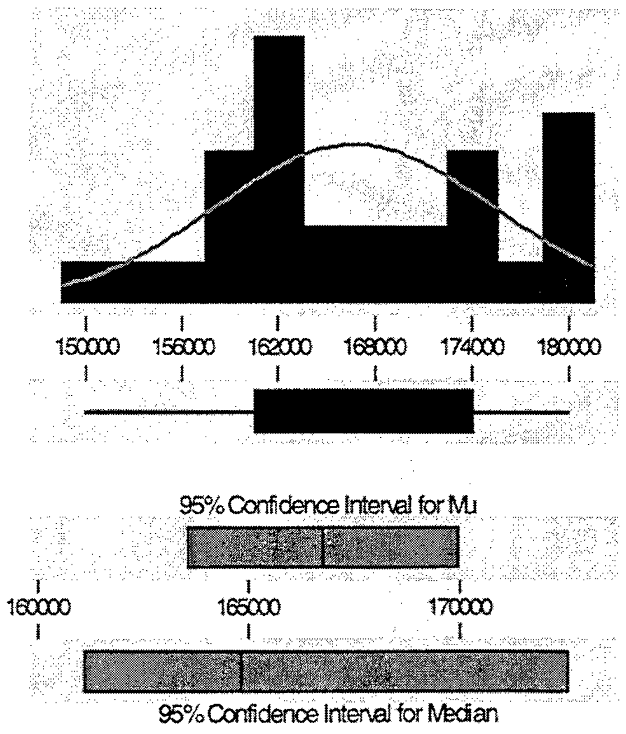
75.3884 77.3701

Descriptive Statistics



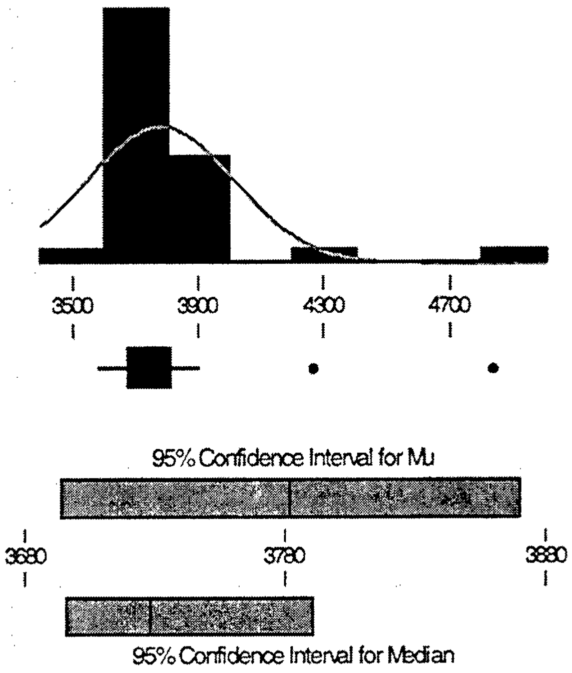
```
Variable:
Detect Wall
Clock Time
Anderson-Darling Normality Test
   A-Squared:          1.219
   P-Value:            0.003

   Mean              93.3564
   StDev              3.2702
   Variance          10.6940
   Skewness          1.28097
   Kurtosis          1.29780
   N                      30

   Minimum            89.135
   1st Quartile       91.299
   Median             92.574
   3rd Quartile       94.672
   Maximum           103.287
95% Confidence Interval for Mu
  92.135              94.578
95% Confidence Interval for Sigma
  2.604                4.396
95% Confidence Interval for Median
  91.614              93.592
```

Descriptive Statistics



```
Variable:  C3I IPC
Bytes Read
Anderson-Darling Normality Test
   A-Squared:          0.698
   P-Value:            0.061

   Mean               166758
   StDev                8602
   Variance         73992428
   Skewness         6.34E-02
   Kurtosis         -1.10990
   N                      30

   Minimum            149864
   1st Quartile       160522
   Median             164812
   3rd Quartile       174002
   Maximum            179904
95% Confidence Interval for Mu
  163546              169970
95% Confidence Interval for Sigma
  6851                 11564
95% Confidence Interval for Median
  161100              172524
```

Descriptive Statistics



Variable: FP IPC Bytes Read

Anderson-Darling Normality Test

A-Squared:	3.569
P-Value:	0.000
Mean	3782.13
StDev	235.80
Variance	55603.6
Skewness	3.14058
Kurtosis	10.7451
N	30
Minimum	3584.00
1st Quartile	3676.00
Median	3728.00
3rd Quartile	3808.00
Maximum	4832.00

95% Confidence Interval for Mu

3694.08 3870.18

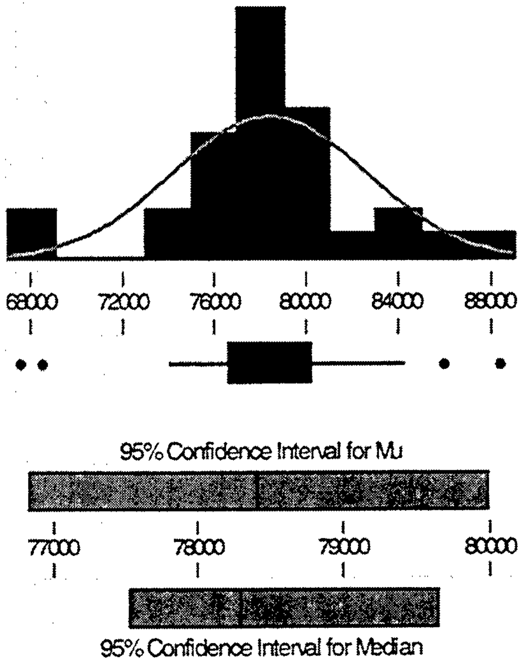
95% Confidence Interval for Sigma

187.80 317.00

95% Confidence Interval for Median

3696.00 3790.85

Descriptive Statistics



Variable: Detect IPC Bytes Read

Anderson-Darling Normality Test

A-Squared:	0.890
P-Value:	0.020
Mean	78411.2
StDev	4229.0
Variance	17884811
Skewness	-2.8E-01
Kurtosis	1.12307
N	30
Minimum	67520.0
1st Quartile	76630.0
Median	78284.0
3rd Quartile	80210.0
Maximum	88352.0

95% Confidence Interval for Mu

76832.0 79990.4

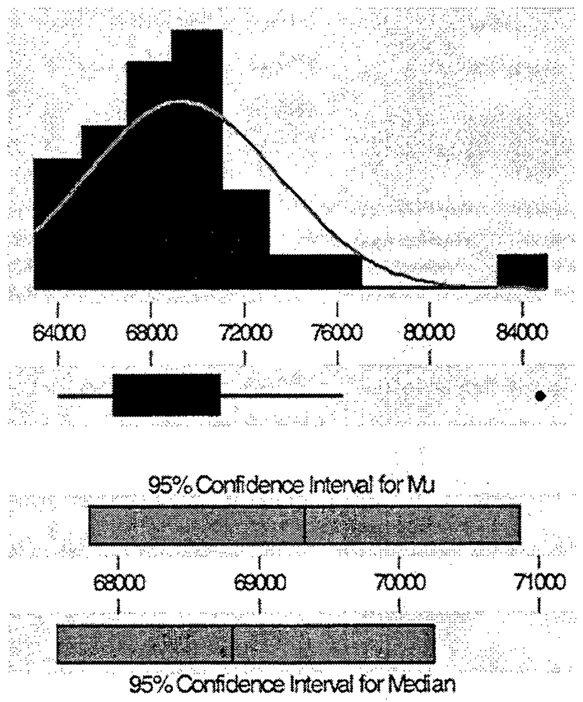
95% Confidence Interval for Sigma

3368.0 5685.2

95% Confidence Interval for Median

77516.1 79646.9

Descriptive Statistics



Variable: C3I IPC Bytes Written

Anderson-Darling Normality Test

A-Squared:	0.870
P-Value:	0.022
Mean	69327.5
StDev	4128.7
Variance	17046254
Skewness	1.64561
Kurtosis	4.14283
N	30
Minimum	64024.0
1st Quartile	66364.0
Median	68792.0
3rd Quartile	70874.0
Maximum	84752.0

95% Confidence Interval for Mu

67785.8	70869.2

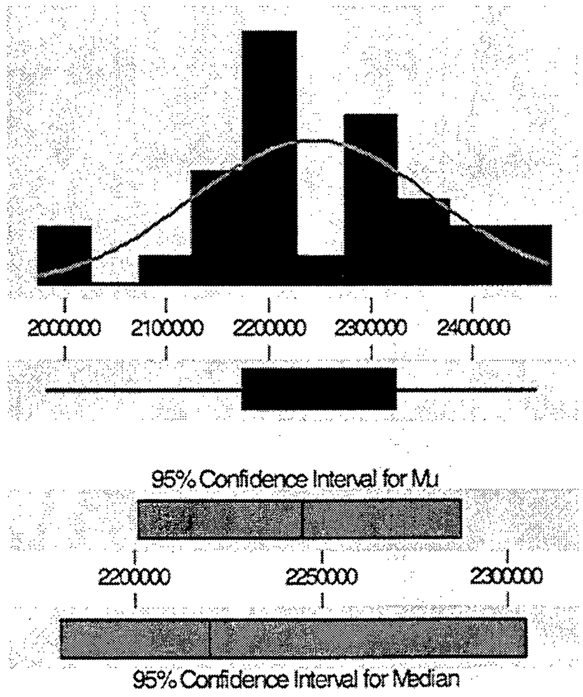
95% Confidence Interval for Sigma

3288.1	5550.3

95% Confidence Interval for Median

67560.3	70250.0

Descriptive Statistics



Variable: FP IPC Bytes Written

Anderson-Darling Normality Test

A-Squared:	0.369
P-Value:	0.404
Mean	2244479
StDev	116675
Variance	1.36E+10
Skewness	-1.2E-01
Kurtosis	-4.7E-01
N	30
Minimum	1982960
1st Quartile	2173564
Median	2219934
3rd Quartile	2322549
Maximum	2460432

95% Confidence Interval for Mu

2200912	2288046

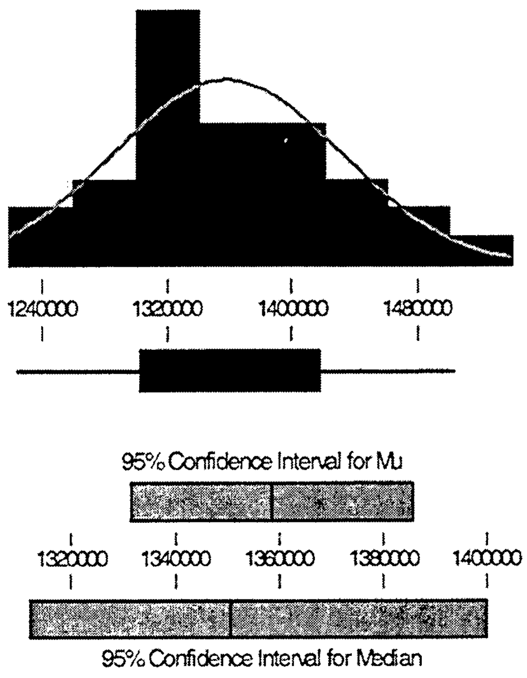
95% Confidence Interval for Sigma

92921	156848

95% Confidence Interval for Median

2180069	2304801

Descriptive Statistics



Variable: Detect IPC Bytes Written

Anderson-Darling Normality Test

A-Squared:	0.301
P-Value:	0.557
Mean	1358588
StDev	73026
Variance	5.33E+09
Skewness	0.215106
Kurtosis	-9.0E-01
N	30
Minimum	1225124
1st Quartile	1302537
Median	1350522
3rd Quartile	1417316
Maximum	1502744

95% Confidence Interval for Mu

1331319 1385856

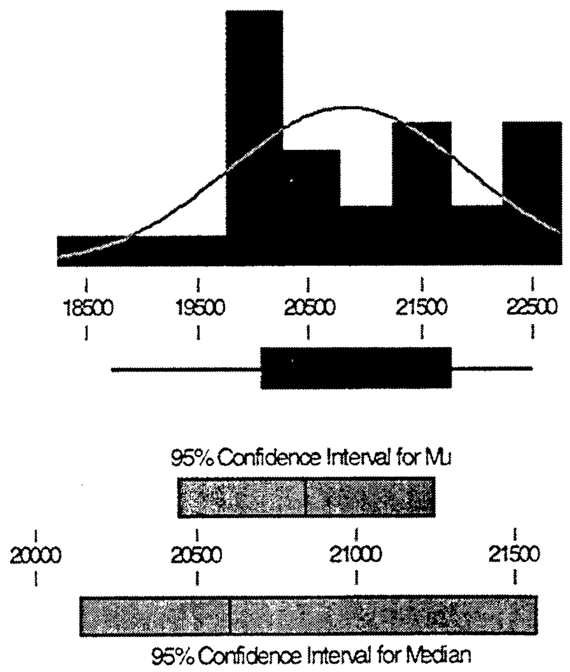
95% Confidence Interval for Sigma

58159 98170

95% Confidence Interval for Median

1312202 1399865

Descriptive Statistics



Variable: C3I IPC Number of Reads

Anderson-Darling Normality Test

A-Squared:	0.698
P-Value:	0.061
Mean	20844.8
StDev	1075.2
Variance	1156132
Skewness	6.34E-02
Kurtosis	-1.10990
N	30
Minimum	18733.0
1st Quartile	20065.3
Median	20601.5
3rd Quartile	21750.3
Maximum	22488.0

95% Confidence Interval for Mu

20443.3 21246.3

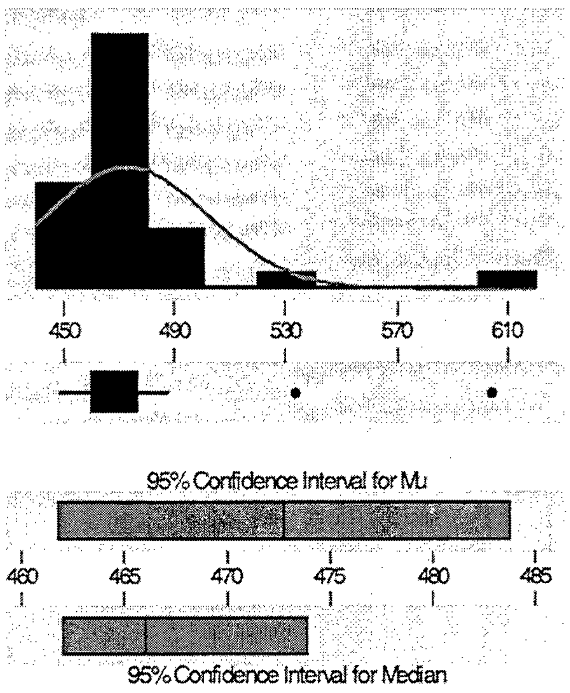
95% Confidence Interval for Sigma

856.3 1445.5

95% Confidence Interval for Median

20137.5 21565.5

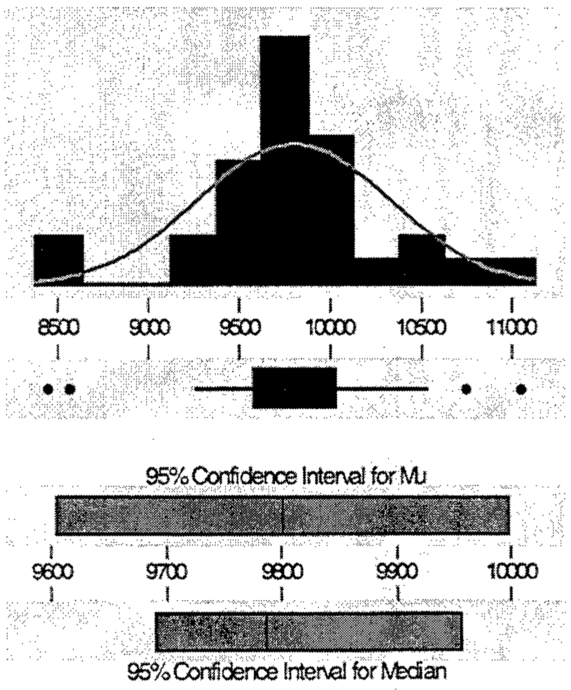
Descriptive Statistics



Variable: FP IPC Number of Reads

Anderson-Darling Normality Test	
A-Squared:	3.569
P-Value:	0.000
Mean	472.767
StDev	29.476
Variance	868.806
Skewness	3.14058
Kurtosis	10.7451
N	30
Minimum	448.000
1st Quartile	459.500
Median	466.000
3rd Quartile	476.000
Maximum	604.000
95% Confidence Interval for Mu	
461.760	483.773
95% Confidence Interval for Sigma	
23.475	39.624
95% Confidence Interval for Median	
462.000	473.856

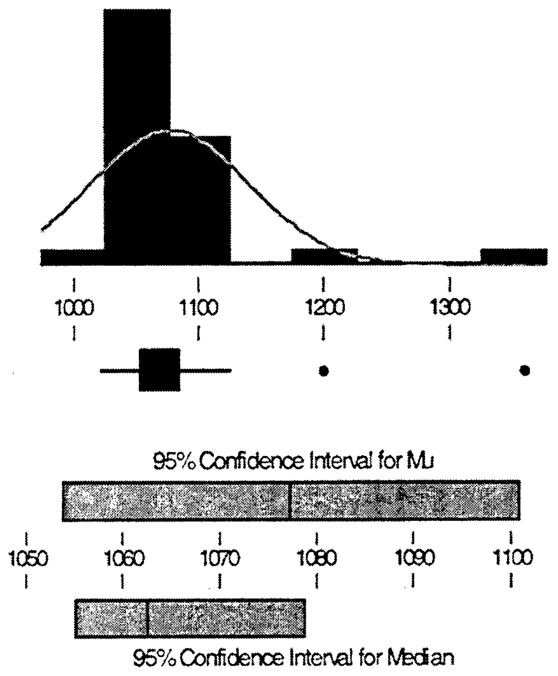
Descriptive Statistics



Variable: Detect IPC Number of Reads

Anderson-Darling Normality Test	
A-Squared:	0.890
P-Value:	0.020
Mean	9801.40
StDev	528.63
Variance	279450
Skewness	-2.8E-01
Kurtosis	1.12307
N	30
Minimum	8440.0
1st Quartile	9578.7
Median	9785.5
3rd Quartile	10026.2
Maximum	11044.0
95% Confidence Interval for Mu	
9604.0	9998.8
95% Confidence Interval for Sigma	
421.0	710.6
95% Confidence Interval for Median	
9689.5	9955.9

Descriptive Statistics



Variable: C3I IPC Number of Writes

Anderson-Darling Normality Test

A-Squared:	3.439
P-Value:	0.000
Mean	1077.30
StDev	63.03
Variance	3972.98
Skewness	3.09582
Kurtosis	10.6610
N	30
Minimum	1022.00
1st Quartile	1052.75
Median	1062.50
3rd Quartile	1083.00
Maximum	1358.00

95% Confidence Interval for Mu

1053.76	1100.84

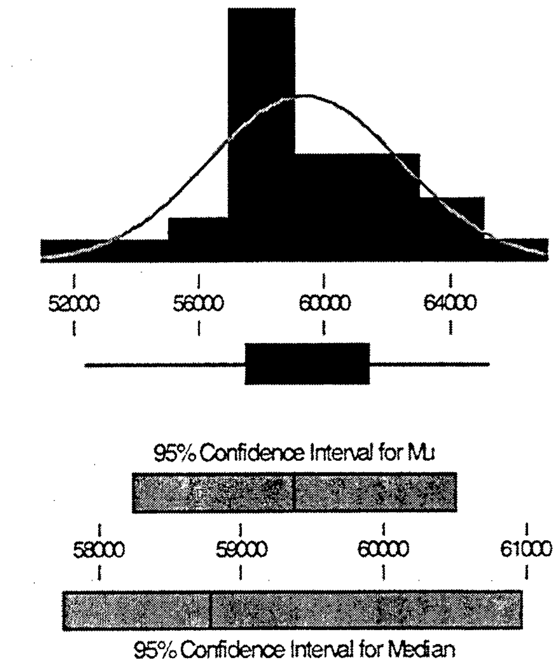
95% Confidence Interval for Sigma

50.20	84.73

95% Confidence Interval for Median

1055.00	1078.77

Descriptive Statistics



Variable: FP IPC Number of Writes

Anderson-Darling Normality Test

A-Squared:	0.353
P-Value:	0.443
Mean	59372.4
StDev	3059.6
Variance	9361193
Skewness	-1.4E-01
Kurtosis	-3.8E-01
N	30
Minimum	52426.0
1st Quartile	57507.0
Median	58784.0
3rd Quartile	61376.0
Maximum	65144.0

95% Confidence Interval for Mu

58229.9	60514.9

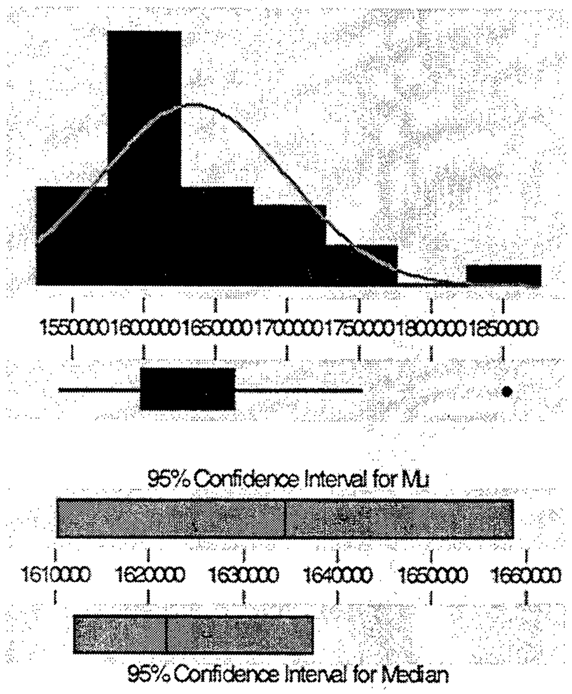
95% Confidence Interval for Sigma

2436.7	4113.1

95% Confidence Interval for Median

57742.9	60958.0

Descriptive Statistics



Variable: C3I Network Bytes Written

Anderson-Darling Normality Test

A-Squared:	1.049
P-Value:	0.008
Mean	1634463
StDev	64789
Variance	4.20E+09
Skewness	1.32862
Kurtosis	2.16355
N	30
Minimum	1541245
1st Quartile	1598278
Median	1621830
3rd Quartile	1661383
Maximum	1851427

95% Confidence Interval for Mu

1610270 1658656

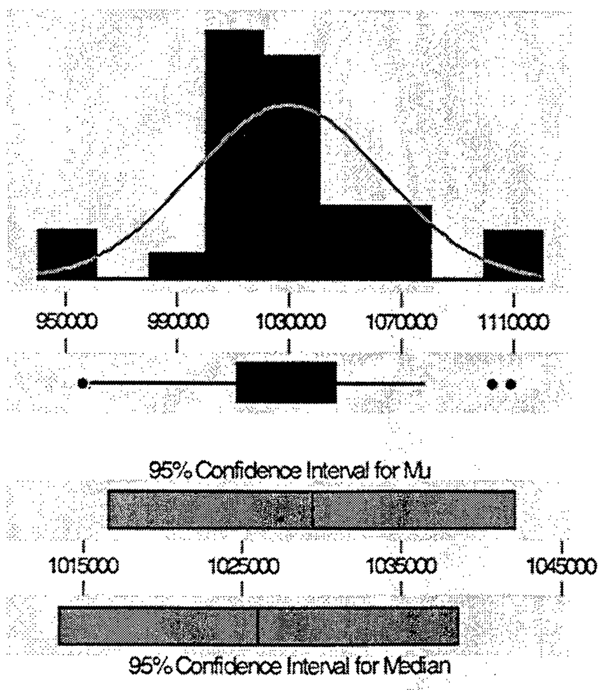
95% Confidence Interval for Sigma

51598 87097

95% Confidence Interval for Median

1611896 1637375

Descriptive Statistics



Variable: FP Network Bytes Written

Anderson-Darling Normality Test

A-Squared:	0.709
P-Value:	0.057
Mean	1029378
StDev	34171
Variance	1.17E+09
Skewness	0.273428
Kurtosis	0.365661
N	30
Minimum	955221
1st Quartile	1011185
Median	1025848
3rd Quartile	1046070
Maximum	1108747

95% Confidence Interval for Mu

1016618 1042137

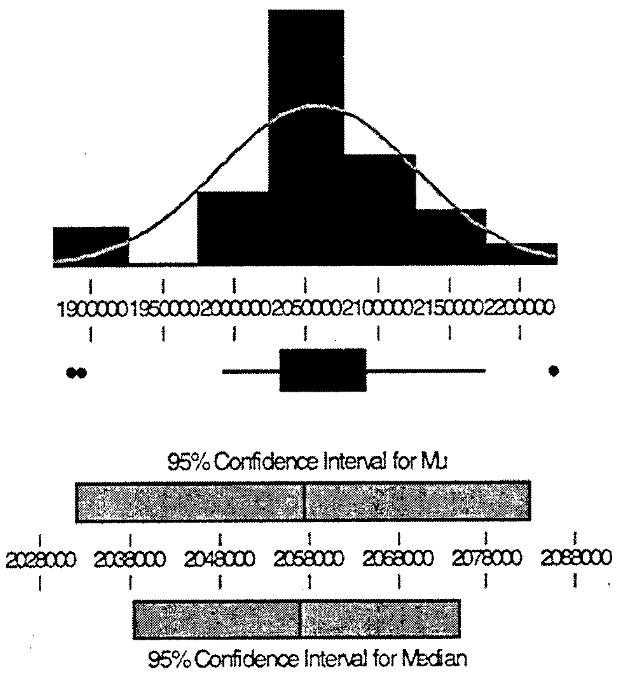
95% Confidence Interval for Sigma

27214 45936

95% Confidence Interval for Median

1013469 1038536

Descriptive Statistics



Variable: Detect Network Bytes Written

Anderson-Darling Normality Test

A-Squared:	0.982
P-Value:	0.012
Mean	2057529
StDev	68161
Variance	4.65E+09
Skewness	-2.3E-01
Kurtosis	1.22879
N	30
Minimum	1885618
1st Quartile	2031778
Median	2056892
3rd Quartile	2090429
Maximum	2222940

95% Confidence Interval for Mu

2032078 2082981

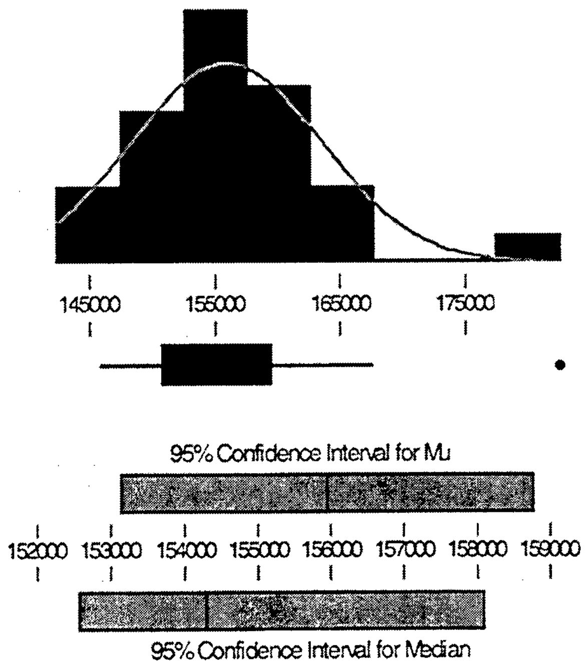
95% Confidence Interval for Sigma

54284 91630

95% Confidence Interval for Median

2038369 2074922

Descriptive Statistics



Variable: C3I Network Number of Writes

Anderson-Darling Normality Test

A-Squared:	0.733
P-Value:	0.050
Mean	155957
StDev	7568
Variance	57276532
Skewness	1.41476
Kurtosis	2.73825
N	30
Minimum	145958
1st Quartile	150646
Median	154280
3rd Quartile	159340
Maximum	182493

95% Confidence Interval for Mu

153131 158783

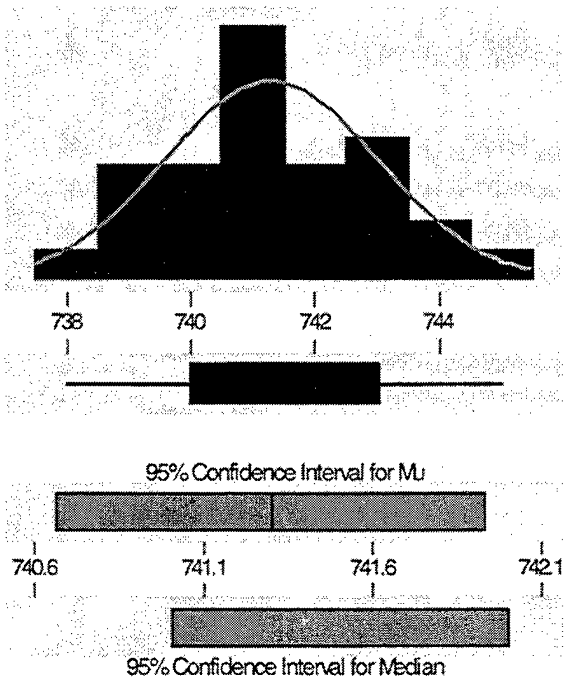
95% Confidence Interval for Sigma

6027 10174

95% Confidence Interval for Median

152574 158076

Descriptive Statistics



Variable: FP Network Number of Writes

Anderson-Darling Normality Test

A-Squared:	0.561
P-Value:	0.134
Mean	741.300
StDev	1.705
Variance	2.90690
Skewness	0.150117
Kurtosis	-7.2E-01
N	30
Minimum	738.000
1st Quartile	740.000
Median	741.000
3rd Quartile	743.000
Maximum	745.000

95% Confidence Interval for Mu

740.663 741.937

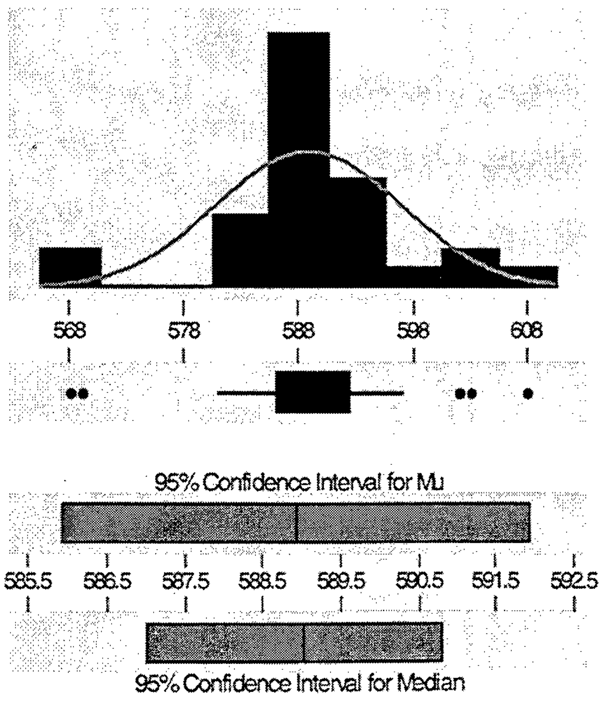
95% Confidence Interval for Sigma

1.358 2.292

95% Confidence Interval for Median

741.000 742.000

Descriptive Statistics



Variable: Detect Network Number of Writes

Anderson-Darling Normality Test

A-Squared:	1.265
P-Value:	0.002
Mean	588.933
StDev	8.034
Variance	64.5471
Skewness	-3.3E-01
Kurtosis	1.49254
N	30
Minimum	568.000
1st Quartile	586.000
Median	589.000
3rd Quartile	592.250
Maximum	608.000

95% Confidence Interval for Mu

585.933 591.933

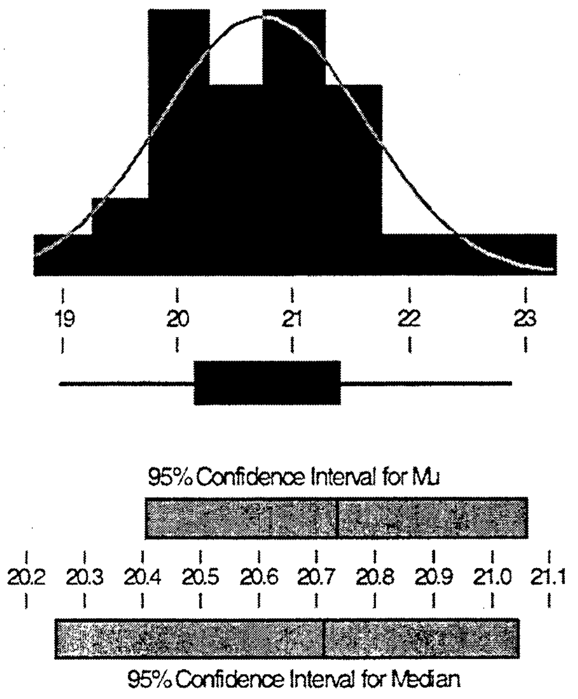
95% Confidence Interval for Sigma

6.398 10.800

95% Confidence Interval for Median

587.000 590.771

Descriptive Statistics



Variable: C3I
Cumulative CPU Times
(c3i.pstat)

Anderson-Darling Normality Test

A-Squared:	0.200
P-Value:	0.871
Mean	20.7333
StDev	0.8770
Variance	0.769195
Skewness	0.324038
Kurtosis	-2.3E-01
N	30
Minimum	18.9700
1st Quartile	20.1500
Median	20.7100
3rd Quartile	21.3925
Maximum	22.8600

95% Confidence Interval for Mu

20.4058 21.0608

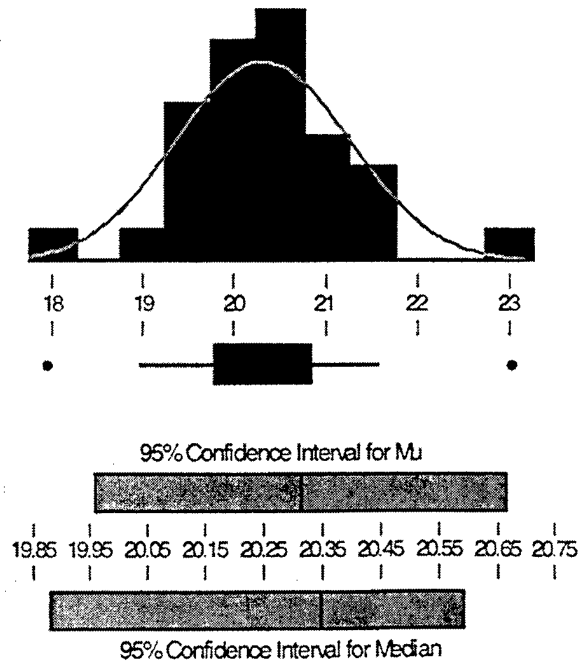
95% Confidence Interval for Sigma

0.6985 1.1790

95% Confidence Interval for Median

20.2491 21.0440

Descriptive Statistics



Variable: FP
Cumulative CPU Times
(fp.pstat)

Anderson-Darling Normality Test

A-Squared:	0.308
P-Value:	0.541
Mean	20.3123
StDev	0.9480
Variance	0.898646
Skewness	0.232861
Kurtosis	1.10592
N	30
Minimum	17.9300
1st Quartile	19.7825
Median	20.3450
3rd Quartile	20.8250
Maximum	23.0200

95% Confidence Interval for Mu

19.9584 20.6663

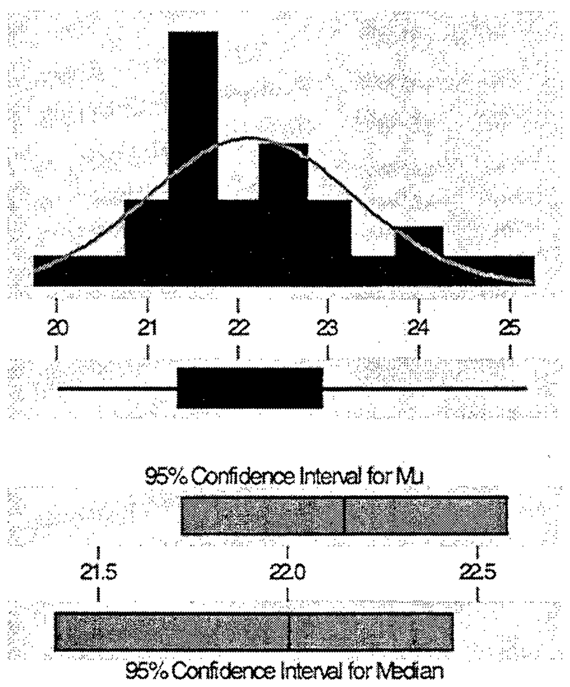
95% Confidence Interval for Sigma

0.7550 1.2744

95% Confidence Interval for Median

19.8800 20.5931

Descriptive Statistics



Variable: Detect Cumulative CPU Times (detect.pstat)

Anderson-Darling Normality Test

A-Squared:	0.575
P-Value:	0.123
Mean	22.1503
StDev	1.1457
Variance	1.31253
Skewness	0.668464
Kurtosis	-2.6E-02
N	30
Minimum	20.0300
1st Quartile	21.3250
Median	22.0000
3rd Quartile	22.9225
Maximum	25.1600

95% Confidence Interval for Mu

21.7225 22.5781

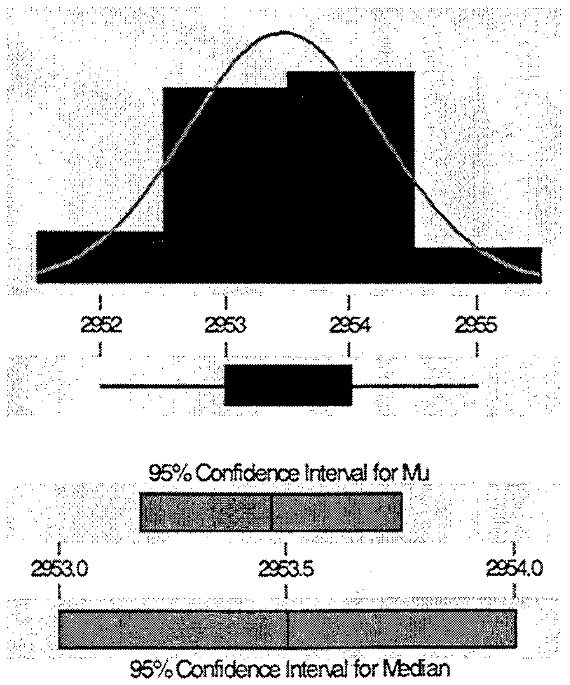
95% Confidence Interval for Sigma

0.9124 1.5401

95% Confidence Interval for Median

21.3869 22.4326

Descriptive Statistics



Variable: C3I Physical Pages in Memory

Anderson-Darling Normality Test

A-Squared:	2.113
P-Value:	0.000
Mean	2953.47
StDev	0.78
Variance	0.602299
Skewness	-1.1E-01
Kurtosis	-5.6E-01
N	30
Minimum	2952.00
1st Quartile	2953.00
Median	2953.50
3rd Quartile	2954.00
Maximum	2955.00

95% Confidence Interval for Mu

2953.18 2953.76

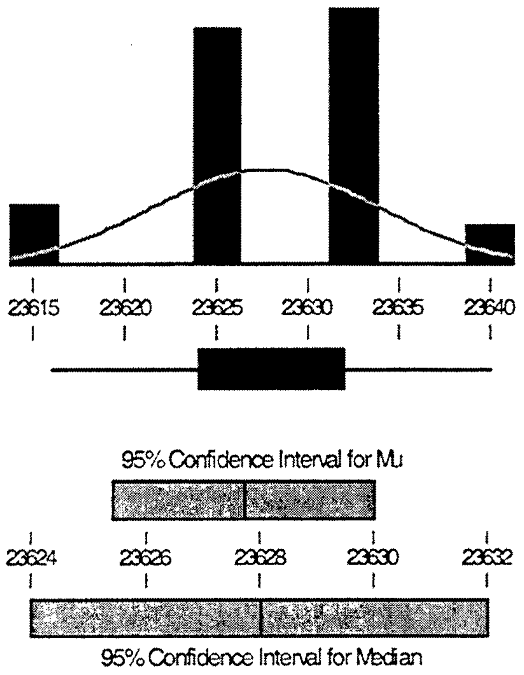
95% Confidence Interval for Sigma

0.62 1.04

95% Confidence Interval for Median

2953.00 2954.00

Descriptive Statistics



Variable: C3I Pages in Virtual Memory

Anderson-Darling Normality Test

A-Squared:	2.113
P-Value:	0.000
Mean	23627.7
StDev	6.2
Variance	38.5471
Skewness	-1.1E-01
Kurtosis	-5.6E-01
N	30
Minimum	23616.0
1st Quartile	23624.0
Median	23628.0
3rd Quartile	23632.0
Maximum	23640.0

95% Confidence Interval for Mu

23625.4 23630.1

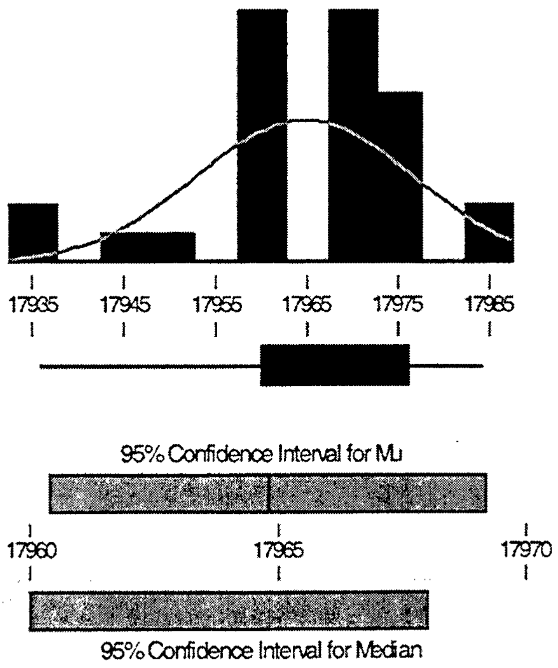
95% Confidence Interval for Sigma

4.9 8.3

95% Confidence Interval for Median

23624.0 23632.0

Descriptive Statistics



Variable: C3I Pages in Resident Set

Anderson-Darling Normality Test

A-Squared:	1.148
P-Value:	0.004
Mean	17964.8
StDev	11.8
Variance	139.476
Skewness	-7.5E-01
Kurtosis	0.361664
N	30
Minimum	17936.0
1st Quartile	17960.0
Median	17968.0
3rd Quartile	17976.0
Maximum	17984.0

95% Confidence Interval for Mu

17960.4 17969.2

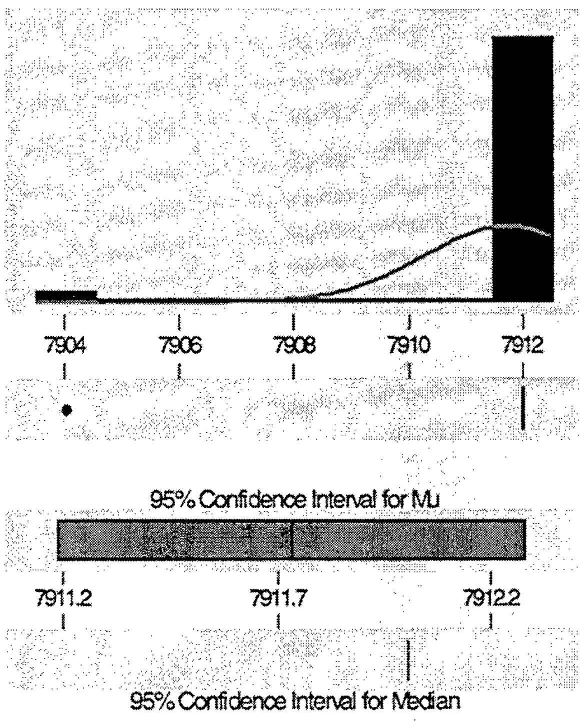
95% Confidence Interval for Sigma

9.4 15.9

95% Confidence Interval for Median

17960.0 17968.0

Descriptive Statistics



Variable: FP Pages in Resident Set

Anderson-Darling Normality Test

A-Squared:	11.090
P-Value:	0.000
Mean	7911.73
StDev	1.46
Variance	2.13333
Skewness	-4.94167
Kurtosis	23.1967
N	30
Minimum	7904.00
1st Quartile	7912.00
Median	7912.00
3rd Quartile	7912.00
Maximum	7912.00

95% Confidence Interval for Mu

7911.19	7912.28

95% Confidence Interval for Sigma

1.16	1.96

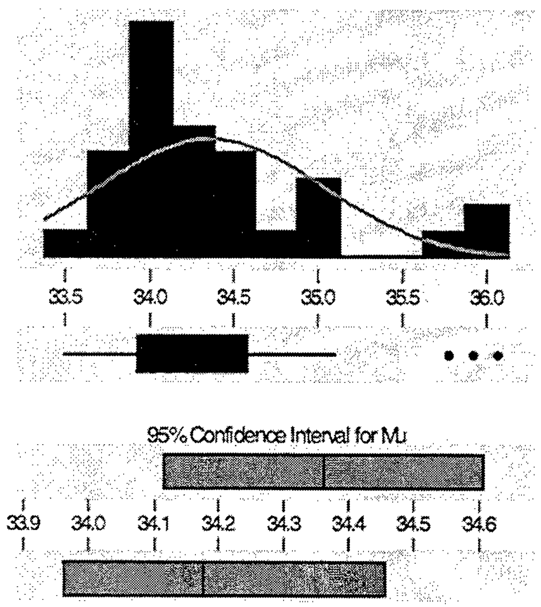
95% Confidence Interval for Median

7912.00	7912.00

APPENDIX I. MSHN WRAPPER OUTPUT: DETERMINISTIC TRIALS

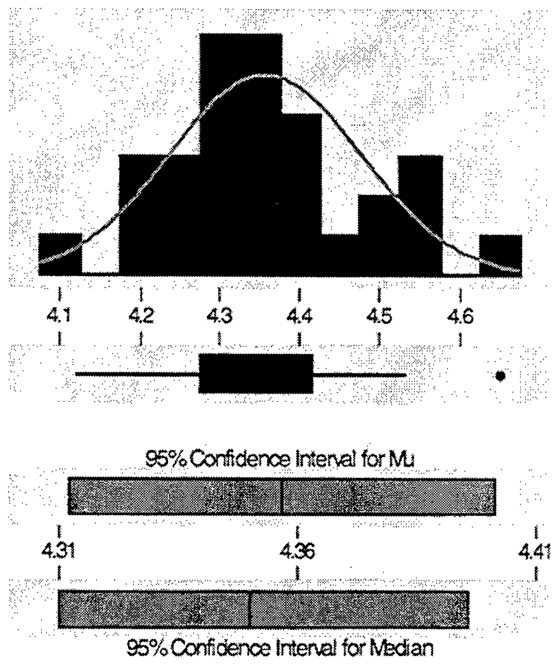
Note: Only data that generated a distribution is displayed.

Descriptive Statistics



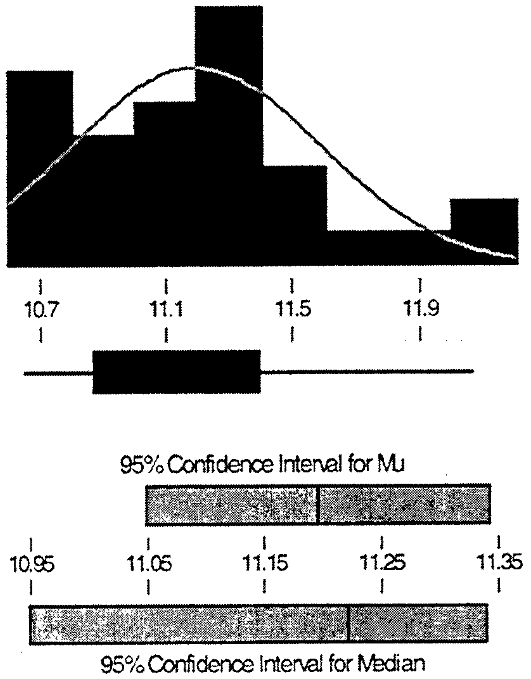
```
Variable: C3I User
CPU Time
(Deterministic)
Anderson-Darling Normality Test
    A-Squared:        1.433
    P-Value:          0.001

    Mean            34.3613
    StDev            0.6598
    Variance       0.435329
    Skewness        1.18541
    Kurtosis       0.564623
    N                    30

    Minimum         33.4900
    1st Quartile    33.9200
    Median          34.1750
    3rd Quartile    34.5775
    Maximum         36.0600
  95% Confidence Interval for Mu
    34.1150         34.6077
 95% Confidence Interval for Sigma
    0.5255          0.8870
 95% Confidence Interval for Median
```

Descriptive Statistics



```
Variable: FP System
CPU Time
(Deterministic)
Anderson-Darling Normality Test
    A-Squared:          0.380
    P-Value:            0.381

    Mean              4.35667
    StDev             0.11989
    Variance         1.44E-02
    Skewness         0.389254
    Kurtosis         -3.3E-01
    N                      30

    Minimum           4.12000
    1st Quartile      4.27500
    Median            4.35000
    3rd Quartile      4.41500
    Maximum           4.65000
  95% Confidence Interval for Mu
    4.31190           4.40144
 95% Confidence Interval for Sigma
    0.09548           0.16118
 95% Confidence Interval for Median
    4.31000           4.39543
```

Descriptive Statistics



Variable: Detect System CPU Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	0.407
P-Value:	0.328
Mean	11.1957
StDev	0.3933
Variance	0.154687
Skewness	0.480020
Kurtosis	-5.8E-01
N	30
Minimum	10.6500
1st Quartile	10.8675
Median	11.2200
3rd Quartile	11.3875
Maximum	12.0600

95% Confidence Interval for Mu

11.0488 11.3425

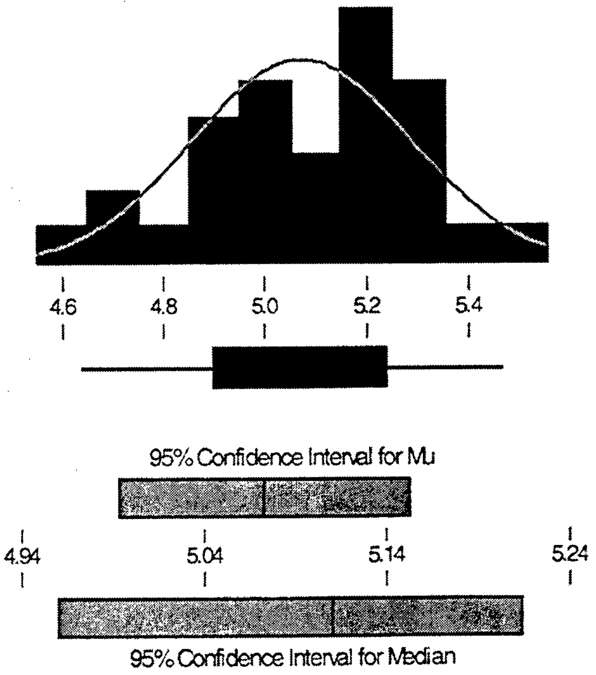
95% Confidence Interval for Sigma

0.3132 0.5287

95% Confidence Interval for Median

10.9483 11.3386

Descriptive Statistics



Variable: C3I System CPU Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	0.393
P-Value:	0.356
Mean	5.07300
StDev	0.21459
Variance	4.60E-02
Skewness	-3.0E-01
Kurtosis	-8.2E-01
N	30
Minimum	4.64000
1st Quartile	4.89750
Median	5.11000
3rd Quartile	5.23500
Maximum	5.46000

95% Confidence Interval for Mu

4.99287 5.15313

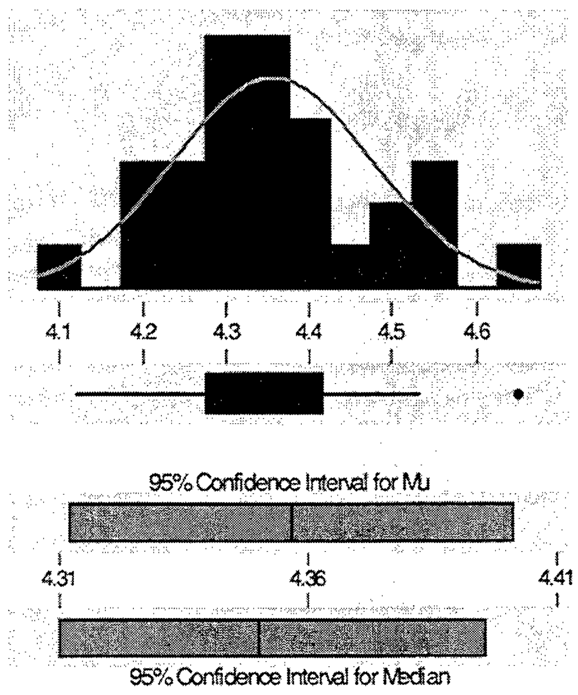
95% Confidence Interval for Sigma

0.17090 0.28848

95% Confidence Interval for Median

4.96000 5.21314

Descriptive Statistics



Variable: FP System CPU Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	0.380
P-Value:	0.381
Mean	4.35667
StDev	0.11989
Variance	1.44E-02
Skewness	0.389254
Kurtosis	-3.3E-01
N	30
Minimum	4.12000
1st Quartile	4.27500
Median	4.35000
3rd Quartile	4.41500
Maximum	4.65000

95% Confidence Interval for Mu

4.31190	4.40144

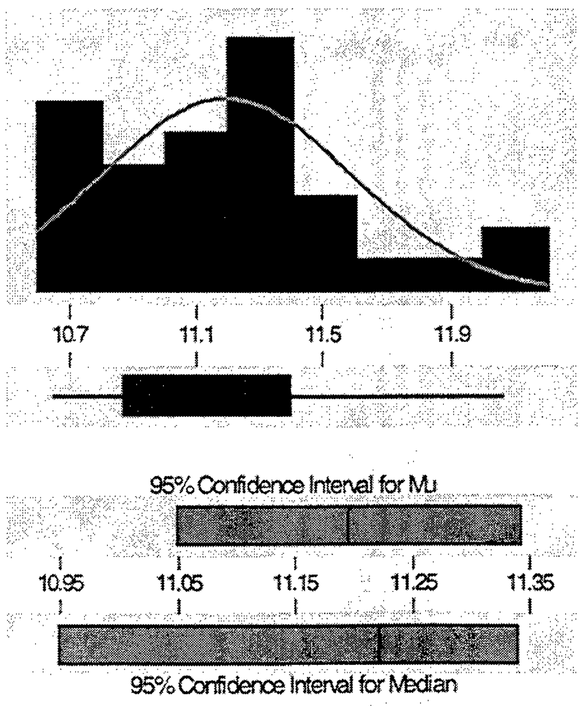
95% Confidence Interval for Sigma

0.09548	0.16118

95% Confidence Interval for Median

4.31000	4.39543

Descriptive Statistics



Variable: Detect System CPU Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	0.407
P-Value:	0.328
Mean	11.1957
StDev	0.3933
Variance	0.154687
Skewness	0.480020
Kurtosis	-5.8E-01
N	30
Minimum	10.6500
1st Quartile	10.8675
Median	11.2200
3rd Quartile	11.3875
Maximum	12.0600

95% Confidence Interval for Mu

11.0488	11.3425

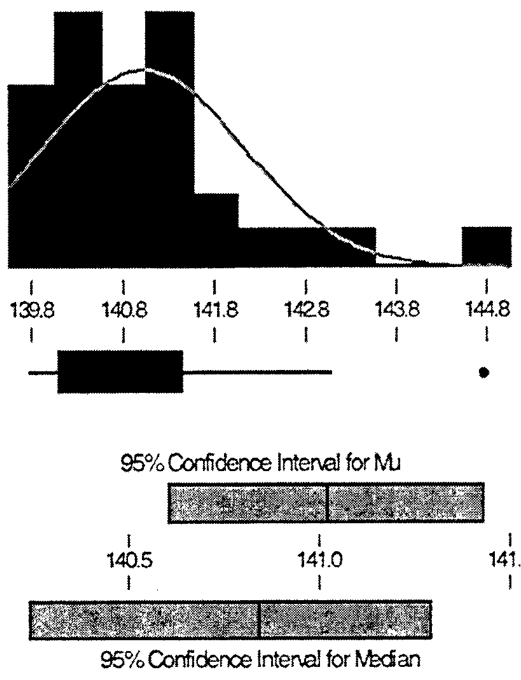
95% Confidence Interval for Sigma

0.3132	0.5287

95% Confidence Interval for Median

10.9483	11.3386

Descriptive Statistics



Variable: C3I Wall Clock Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	1.104
P-Value:	0.006
Mean	141.020
StDev	1.107
Variance	1.22540
Skewness	1.45440
Kurtosis	2.30142
N	30
Minimum	139.770
1st Quartile	140.099
Median	140.840
3rd Quartile	141.416
Maximum	144.761

95% Confidence Interval for Mu

140.606	141.433

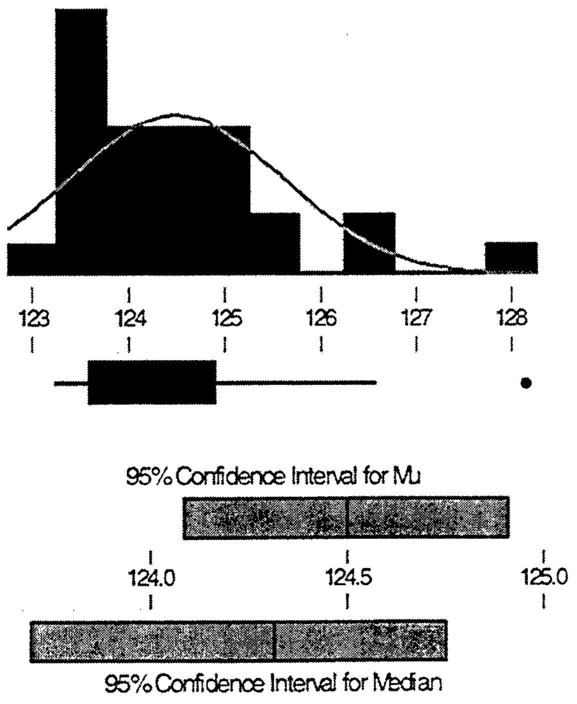
95% Confidence Interval for Sigma

0.882	1.488

95% Confidence Interval for Median

140.247	141.289

Descriptive Statistics



Variable: FP Wall Clock Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	1.050
P-Value:	0.008
Mean	124.497
StDev	1.108
Variance	1.22803
Skewness	1.35971
Kurtosis	1.89990
N	30
Minimum	123.244
1st Quartile	123.584
Median	124.309
3rd Quartile	124.885
Maximum	128.139

95% Confidence Interval for Mu

124.083	124.911

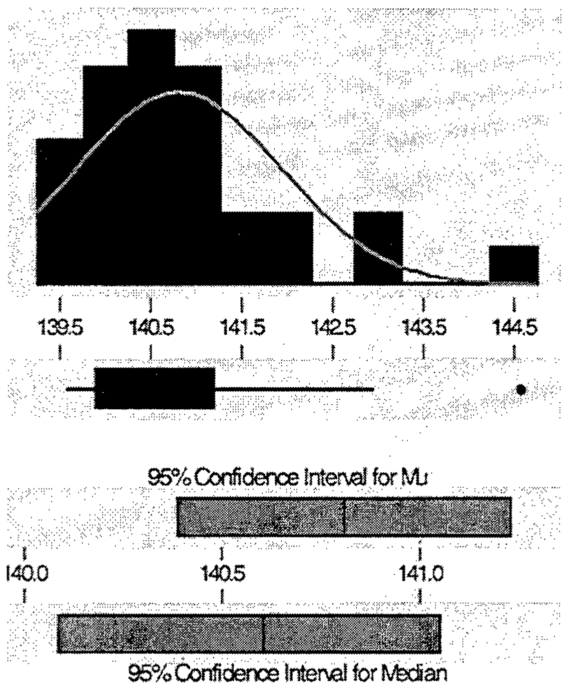
95% Confidence Interval for Sigma

0.883	1.490

95% Confidence Interval for Median

123.698	124.746

Descriptive Statistics



Variable: Detect Wall Clock Time (Deterministic)

Anderson-Darling Normality Test

A-Squared:	1.120
P-Value:	0.005
Mean	140.808
StDev	1.130
Variance	1.27736
Skewness	1.43378
Kurtosis	2.12833
N	30
Minimum	139.579
1st Quartile	139.885
Median	140.600
3rd Quartile	141.175
Maximum	144.573

95% Confidence Interval for Mu

140.386 141.230

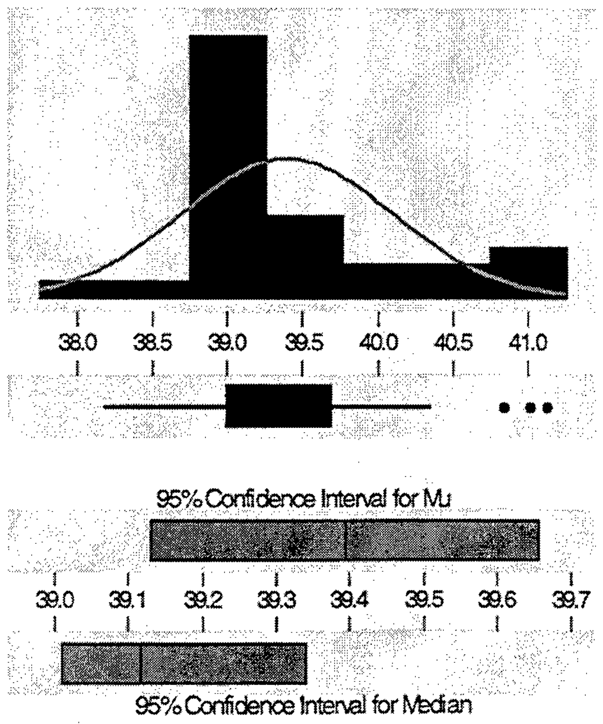
95% Confidence Interval for Sigma

0.900 1.519

95% Confidence Interval for Median

140.085 141.046

Descriptive Statistics



Variable: C3I Cumulative CPU Time (c3i.pstat)

Anderson-Darling Normality Test

A-Squared:	2.236
P-Value:	0.000
Mean	39.3937
StDev	0.7044
Variance	0.496127
Skewness	1.07294
Kurtosis	0.315679
N	30
Minimum	38.1800
1st Quartile	38.9925
Median	39.1150
3rd Quartile	39.6700
Maximum	41.1200

95% Confidence Interval for Mu

39.1307 39.6567

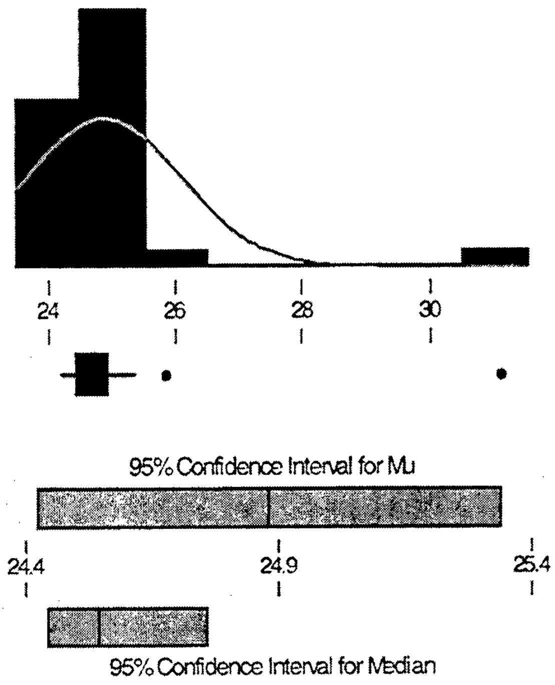
95% Confidence Interval for Sigma

0.5610 0.9469

95% Confidence Interval for Median

39.0100 39.3400

Descriptive Statistics



Variable: FP
Cumulative CPU Time
(fp.pstat)

Anderson-Darling Normality Test

A-Squared:	5.549
P-Value:	0.000
Mean	24.8813
StDev	1.2281
Variance	1.50816
Skewness	4.29112
Kurtosis	18.8038
N	30
Minimum	24.2200
1st Quartile	24.4250
Median	24.5400
3rd Quartile	24.8925
Maximum	31.0900

95% Confidence Interval for Mu

24.4228 25.3399

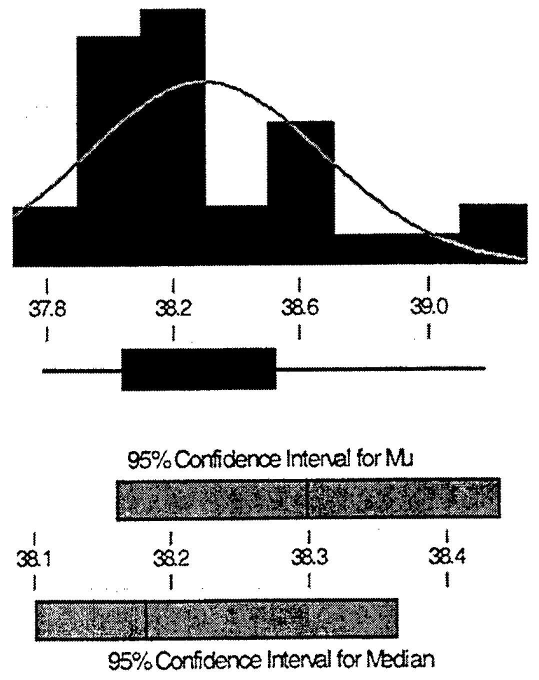
95% Confidence Interval for Sigma

0.9780 1.6509

95% Confidence Interval for Median

24.4423 24.7577

Descriptive Statistics



Variable: Detect
Cumulative CPU Time
(detect.pstat)

Anderson-Darling Normality Test

A-Squared:	1.014
P-Value:	0.010
Mean	38.2990
StDev	0.3719
Variance	0.138306
Skewness	0.907432
Kurtosis	-1.1E-01
N	30
Minimum	37.7900
1st Quartile	38.0375
Median	38.1800
3rd Quartile	38.5175
Maximum	39.1700

95% Confidence Interval for Mu

38.1601 38.4379

95% Confidence Interval for Sigma

0.2962 0.4999

95% Confidence Interval for Median

38.1014 38.3631

APPENDIX J. PAPA1 REPORT

```
!                                    PAPA Report
!                                    -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK      Single Shot PK
!       PKCum     Cumulative PK Of Individual Shots
!       Route PK  Total PK For Killing Target
!
!    Time             Target                     Engager              Range      Az
El      SSPK   PKCum   Route
! (Seconds)           ID                         ID                    (m)      (Deg)
(Deg)                   PK
!------------  ------------------------- ------------------------- --------- ------- ----
--- ------- ------- -------
       104.96  Missile 28767        Asset_Def_SAM_03             17272   30.60   30.95
80.00   80.00   80.00
       105.52  Missile 28766        Asset_Def_SAM_01             18410   12.98   23.34
80.00   80.00   80.00
       106.89  Missile 28768        Asset_Def_SAM_01             19006    8.33   24.25
80.00   80.00   80.00
       107.83  Missile 28767        Asset_Def_SAM_03             15669   32.85   31.13
80.00   96.00   96.00
       108.03  Missile 28766        Asset_Def_SAM_01             16819   14.03   22.54
80.00   96.00   96.00
       109.74  Missile 28768        Asset_Def_SAM_01             17400    9.64   23.61
80.00   96.00   96.00
       110.42  Hostile_AA_Ftr_04    Point_Def_SAM_01/03           4698    7.43   24.64
50.00   50.00   50.00
       118.45  Hostile_AA_Ftr_04    Point_Def_SAM_01/03           2621   15.19   48.13
50.00   75.00   75.00
       119.54  Hostile_AA_Ftr_06    Point_Def_SAM_01/03           3912  357.86   29.11
50.00   50.00   50.00
       120.30  Missile 28767        Asset_Def_SAM_03              8526   51.85   24.06
80.00   99.20   99.20
       121.18  Missile 28768        Asset_Def_SAM_03              7746   47.40   25.74
80.00   99.20   99.20
       125.23  Hostile_AA_Ftr_05    Point_Def_SAM_01/03           4161   76.62   27.29
50.00   50.00   50.00
       127.88  Hostile_AA_Ftr_06    Point_Def_SAM_01/03           2087  354.08   65.24
50.00   75.00   75.00
       128.33  Hostile_AA_Ftr_04    Point_Def_SAM_01/03           2156  157.01   64.40
50.00   87.50   87.50
       132.40  Missile 28773        Asset_Def_SAM_01             18410   12.98   23.34
80.00   80.00   80.00
       132.48  Missile 28774        Asset_Def_SAM_03             15669   32.85   31.13
80.00   80.00   80.00
       134.01  Missile 28775        Asset_Def_SAM_01             19006    8.33   24.25
80.00   80.00   80.00
       134.47  Hostile_AA_Ftr_07    Point_Def_SAM_01/03           3823    9.04   30.77
50.00   50.00   50.00
       135.11  Missile 28773        Asset_Def_SAM_01             16819   14.03   22.54
80.00   96.00   96.00
       135.19  Missile 28774        Asset_Def_SAM_03             14106   35.50   30.98
80.00   96.00   96.00
       136.82  Missile 28775        Asset_Def_SAM_01             17400    9.64   23.61
80.00   96.00   96.00
       142.93  Hostile_AA_Ftr_07    Point_Def_SAM_01/03           2113   31.15   67.29
50.00   75.00   75.00
       147.77  Missile 28775        Asset_Def_SAM_03              7746   47.40   25.74
80.00   99.20   99.20
```

```
!
!                                       PAPA Report
!                                       -----------
!
!     Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!     Report Type:  PK HISTORY
!
!     PAPA PK Legend
!     --------------
!      SSPK       Single Shot PK
!      PKCum      Cumulative PK Of Individual Shots
!      Route PK   Total PK For Killing Target
!
!   Time               Target                      Engager               Range      Az
El     SSPK   PKCum   Route
! (Seconds)            ID                          ID                    (m)      (Deg)
(Deg)                  PK
!------------  ------------------------- ------------------------- --------- ------- ----
--- -------- ------- -------
             155.10  Red_AG_Ftr_01/04      Point_Def_SAM_01/01          3851  115.99    0.43
50.00    50.00   50.00
             155.53  Red_AG_Ftr_01/03      Point_Def_SAM_01/01          4210  113.46    0.82
50.00    50.00   50.00
             156.97  Red_AG_Ftr_01/01      Point_Def_SAM_01/01          5636  115.14    0.40
50.00    50.00   50.00
             158.05  Hostile_AA_Ftr_02/01  Point_Def_SAM_01/03          6901   37.19   15.52
42.16    42.16   42.16
             158.13  Hostile_AA_Ftr_02/03  Point_Def_SAM_01/03          6980   42.32   16.20
40.93    40.93   40.93
             158.13  Hostile_AA_Ftr_02/04  Point_Def_SAM_01/03          6980   42.32   16.20
40.93    40.93   40.93
             161.66  Missile 28777         Asset_Def_SAM_03            17272   30.60   30.95
80.00    80.00   80.00
             162.51  Missile 28776         Asset_Def_SAM_01            18410   12.98   23.34
80.00    80.00   80.00
             163.89  Missile 28778         Asset_Def_SAM_01            19006    8.33   24.25
80.00    80.00   80.00
             164.56  Missile 28777         Asset_Def_SAM_03            15669   32.85   31.13
80.00    96.00   96.00
             165.03  Missile 28776         Asset_Def_SAM_01            16819   14.03   22.54
80.00    96.00   96.00
             165.70  Red_AG_Ftr_02/03      Point_Def_SAM_01/01          4513  120.74    0.98
50.00    50.00   50.00
             166.62  Red_AG_Ftr_01/04      Point_Def_SAM_01/01          5215  137.52    1.42
50.00    75.00   75.00
             166.74  Missile 28778         Asset_Def_SAM_01            17400    9.64   23.61
80.00    96.00   96.00

166.97  Red_AG_Ftr_01/03              Point_Def_SAM_01/01          5484  134.33    1.72
50.00    75.00   75.00
             169.44  Hostile_AA_Ftr_05     AA_Ftr_04/01                 9129  324.81  354.23
70.00    85.00   85.00
             170.07  Hostile_AA_Ftr_05     AA_Ftr_04/02                 9305  325.33  352.77
70.00    95.50   95.50
             170.41  Hostile_AA_Ftr_02/01  Point_Def_SAM_01/03          4576   38.66   23.85
50.00    71.08   71.08
             170.53  Hostile_AA_Ftr_02/03  Point_Def_SAM_01/03          4693   46.71   24.55
50.00    70.47   70.47
             170.53  Hostile_AA_Ftr_02/04  Point_Def_SAM_01/03          4693   46.71   24.55
50.00    70.47   70.47
             171.83  AA_Ftr_04/01          Hostile_AA_Ftr_05            9394  133.61    3.72
45.00    45.00   45.00
             173.49  AA_Ftr_04/01          Hostile_AA_Ftr_04            7910  138.02  354.45
45.00    69.75   48.09
             176.86  Missile 28777         Asset_Def_SAM_03             9782   46.69   27.23
80.00    99.20   99.20
             177.92  Missile 28778         Asset_Def_SAM_03             7746   47.40   25.74
80.00    99.20   99.20
      !
```

```
!                                  PAPA Report
!                                  -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!    Time              Target                   Engager              Range     Az
El      SSPK    PKCum   Route
! (Seconds)             ID                      ID                   (m)      (Deg)
(Deg)                    PK
!------------  ------------------------ ------------------------- --------- ------- ----
--- ------- ------- -------
       178.41  AA_Ftr_04/02          Hostile_AA_Ftr_04            5762  131.28  350.69
45.00   45.00   45.00
       178.62  Red_AG_Ftr_01/04      Point_Def_SAM_01/01          6992  149.19    0.94
40.75   85.19   85.19
      178.88  Hostile_AA_Ftr_02/02    Point_Def_SAM_01/03          3052   27.07
39.78   50.00   50.00   50.00
      180.18  AA_Ftr_04/01            Hostile_AA_Ftr_06            7440  124.83
349.07   45.00   83.36   53.93
      183.18  Red_AG_Ftr_02/03        Point_Def_SAM_01/01          6818  144.39
1.31    43.46   71.73   71.73
      183.32  AA_Ftr_04/02            Hostile_AA_Ftr_06            6232  130.48
346.44   45.00   69.75   51.19
      187.82  AA_Ftr_04/01            Hostile_AA_Ftr_07            9445  129.68
354.04   45.00   90.85   59.12
      191.44  AA_Ftr_04/02            Hostile_AA_Ftr_07            9307  137.65
356.55   45.00   83.36   56.68
      192.35  Missile 28785           Asset_Def_SAM_03            15669   32.85
31.13   80.00   80.00   80.00
      192.48  Missile 28784           Asset_Def_SAM_01            18410   12.98
23.34   80.00   80.00   80.00
      193.86  Missile 28786           Asset_Def_SAM_01            19006    8.33
24.25   80.00   80.00   80.00
      195.01  Missile 28784           Asset_Def_SAM_01            16819   14.03
22.54   80.00   96.00   96.00
      195.10  Missile 28785           Asset_Def_SAM_03            14106   35.50
30.98   80.00   96.00   96.00
      195.90  Red_CM_01               AA_Ftr_03/02                 9469  349.78
347.20   70.00   70.00   70.00
      196.91  Missile 28786           Asset_Def_SAM_01            17400    9.64
23.61   80.00   96.00   96.00
      197.51  Hostile_AA_Ftr_04       AA_Ftr_03/01                10731   17.30
344.55   70.00   96.25   96.25
      198.57  Hostile_AA_Ftr_04       AA_Ftr_05/01                17463  291.82
349.52   58.93   98.46   98.46
      205.73  AA_Ftr_05/02            Hostile_AA_Ftr_06            6192  101.87
358.92   45.00   45.00   45.00
      208.50  Missile 28786           Asset_Def_SAM_03             7746   47.40
25.74   80.00   99.20   99.20
      209.11  Hostile_AA_Ftr_03/03    Point_Def_SAM_01/02          7640  287.85
16.56   30.61   30.61   30.61
      209.11  Hostile_AA_Ftr_03/04    Point_Def_SAM_01/02          7640  287.85
16.56   30.61   30.61   30.61
      209.23  Red_CM_01               AA_Ftr_03/01                 3150  328.00
320.89   70.00   91.00   91.00
      211.48  Hostile_AA_Ftr_07       AA_Ftr_04/01                 7529  356.22
359.50   70.00   92.50   82.15
      217.46  Red_CM_03               AA_Ftr_03/02                 6644  298.96
342.16   70.00   70.00   70.00
      217.53  Red_CM_04               AA_Ftr_03/02                 4367  287.11
331.46   70.00   70.00   70.00
```

```
!
!                                       PAPA Report
!                                       -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK      Single Shot PK
!       PKCum     Cumulative PK Of Individual Shots
!       Route PK  Total PK For Killing Target
!
!   Time               Target                    Engager             Range      Az
El     SSPK   PKCum   Route
! (Seconds)           ID                         ID                  (m)      (Deg)
(Deg)                  PK
!------------  ------------------------ ------------------------ --------- ------- ----
--- ------- ------- -------
       220.41  Hostile_AA_Ftr_06       AA_Ftr_05/01                   7929   32.30
341.30   70.00   92.50   92.50
       220.86  AA_Ftr_03/01            Hostile_AA_Ftr_04              6741  202.05
14.29   45.00   45.00   45.00
       221.25  Missile 28787           Asset_Def_SAM_01              20044   12.04
23.79   80.00   80.00   80.00
       221.43  AA_Ftr_04/01            Hostile_AA_Ftr_05              6510   61.27
359.08   45.00   94.97   59.94
       221.76  Missile 28788           Asset_Def_SAM_03              17272   30.60
30.95   80.00   80.00   80.00
       222.96  Missile 28789           Asset_Def_SAM_01              19006    8.33
24.25   80.00   80.00   80.00
       223.42  Hostile_AA_Ftr_03/03    Point_Def_SAM_01/02            7850  257.65
16.11   27.34   49.58   49.58
       223.42  Hostile_AA_Ftr_03/04    Point_Def_SAM_01/02            7850  257.65
16.11   27.34   49.58   49.58
      224.18  AA_Ftr_04/02             Hostile_AA_Ftr_05              9667   51.33
359.71   45.00   90.85   57.56
       224.28  Missile 28787           Asset_Def_SAM_01              18410   12.98
23.34   80.00   96.00   96.00
       224.31  Hostile_SAM_Killer_01   Asset_Def_SAM_03              44459  357.04
1.10   80.00   80.00   80.00
       224.67  Missile 28788           Asset_Def_SAM_03              15669   32.85
31.13   80.00   96.00   96.00
       225.85  Missile 28789           Asset_Def_SAM_01              17400    9.64
23.61   80.00   96.00   96.00
       225.90  Red_CM_02               AA_Ftr_03/02                   7669  267.80
344.93   70.00   70.00   70.00
       226.13  AA_Ftr_04/01            Hostile_AA_Ftr_03/01           9211  171.94
348.90   45.00   97.23   77.97
       227.17  AA_Ftr_04/01            Hostile_AA_Ftr_03/02           9391  170.14
348.47   45.00   98.48   87.88
       227.27  AA_Ftr_04/01            Hostile_AA_Ftr_03/03           9315  177.02
348.40   45.00   99.16   90.63
       227.27  AA_Ftr_04/01            Hostile_AA_Ftr_03/04           9315  177.02
348.40   45.00   99.54   92.76
       231.64  Hostile_AA_Ftr_05       AA_Ftr_05/01                  12238  298.84
344.61   70.00   98.65   98.65
       235.47  Hostile_AA_Ftr_04       AA_Ftr_03/02                   5829    9.17
336.67   70.00   99.54   99.54
       235.48  Missile 28787           Asset_Def_SAM_03               8617   46.41
26.40   80.00   99.20   99.20
       237.93  Missile 28788           Asset_Def_SAM_03               8526   51.85
24.06   80.00   99.20   99.20
       238.82  Hostile_AA_Ftr_07       AA_Ftr_05/01                  12391   16.61
345.15   70.00   97.75   94.65
       244.37  Hostile_AA_Ftr_05       AA_Ftr_04/02                   2989  142.77
358.55   70.00   99.60   99.05
```

```
!
!                              PAPA Report
!                              -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!    Time               Target                   Engager              Range      Az
El      SSPK    PKCum   Route
! (Seconds)              ID                      ID                   (m)      (Deg)
(Deg)                     PK
!------------  ------------------------- ------------------------- --------- -------- ----
--- ------- ------- -------
       245.52  AA_Ftr_03/01              Hostile_AA_Ftr_02/01           8986  217.36
350.12   45.00   69.75   52.16
       246.50  AA_Ftr_03/01              Hostile_AA_Ftr_02/02           8399  219.26
348.39   45.00   83.36   62.92
       246.75  AA_Ftr_03/01              Hostile_AA_Ftr_02/03           8311  226.29
348.25   45.00   90.85   67.85
       246.75  AA_Ftr_03/01              Hostile_AA_Ftr_02/04           8311  226.29
348.25   45.00   94.97   72.12
       250.74  Red_CM_01                 Escort_02/01                  10273    8.81
294.46   70.00   97.30   97.30
       251.27  Red_CM_01                 Escort_02/02                  10763    5.72
298.60   70.00   99.19   99.19
       256.51  Hostile_AA_Ftr_07         AA_Ftr_05/02                   3634   58.93
341.68   70.00   99.33   96.71
       256.92  Hostile_AA_Ftr_06         AA_Ftr_05/02                  14751  169.76
355.94   64.32   97.32   95.15
       262.99  Hostile_AA_Ftr_03/02      AA_Ftr_04/01                   7858  100.21
7.76    70.00   70.00   70.00
       264.00  Hostile_SAM_Killer_01     Asset_Def_SAM_03              35534  355.11
1.45    80.00   96.00   96.00
       265.29  Red_AG_Ftr_03             Asset_Def_SAM_03              46934   11.59
0.61    80.00   80.00   80.00
       268.59  Hostile_AA_Ftr_05         AA_Ftr_04/01                   6558   44.80
359.92   70.00   99.88   99.10
       275.52  Red_AG_Ftr_01/04          AA_Ftr_05/01                   4908  103.91
323.74   70.00   95.56   95.56
       278.50  Hostile_AA_Ftr_03/01      AA_Ftr_04/01                   6849  135.54
13.51    70.00   70.00   70.00
       279.36  Red_AG_Ftr_02/04          AA_Ftr_05/01                   3851   89.07
311.50   70.00   70.00   70.00
       279.37  Red_CM_04                 Escort_02/02                  12471  270.30
312.51   70.00   91.00   91.00
       279.95  Hostile_AA_Ftr_02/03      Asset_Def_SAM_03              51330   34.29
2.25    80.00   94.09   94.09
       280.93  AA_Ftr_03/01              Hostile_AA_Ftr_04              2886  307.73
359.49   45.00   97.23   72.18
       281.50  Red_CM_04                 Escort_02/01                  12573  266.47
313.65   70.00   97.30   97.30
       281.76  Hostile_AA_Ftr_05         Point_Def_SAM_01/03            5796   85.56
1.10    50.00   99.94   99.55
       282.45  Hostile_AA_Ftr_02/04      Asset_Def_SAM_03              51450   35.42
2.58    80.00   94.09   94.09
       282.56  AA_Ftr_03/02              Hostile_AA_Ftr_02/01          11582  121.10
6.41    45.00   45.00   45.00
       282.78  Hostile_AA_Ftr_04         AA_Ftr_03/01                   2316  115.20
2.04    70.00   99.86   99.63
       283.45  Red_AG_Ftr_02/03          AA_Ftr_05/01                   3644   95.06
309.17   70.00   91.52   91.52
```

```
!
!                                 PAPA Report
!                                 -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!    Time               Target                    Engager               Range      Az
El      SSPK    PKCum   Route
! (Seconds)             ID                        ID                    (m)      (Deg)
(Deg)                    PK
!------------  ------------------------ ------------------------ --------- ------- ----
--- ------- ------- -------
       288.76  AA_Ftr_05/02            Hostile_AA_Ftr_01/03          9409  240.06
279.88   45.00   69.75   69.75
       288.76  AA_Ftr_05/02            Hostile_AA_Ftr_01/04          9409  240.06
279.88   45.00   83.36   83.36
       288.85  AA_Ftr_05/02            Hostile_AA_Ftr_01/01          9355  249.17
281.41   45.00   90.85   90.85

       288.88  AA_Ftr_05/02           Hostile_AA_Ftr_01/02          9304  215.63  274.89
45.00   94.97   94.97
       289.93  AA_Ftr_04/01            Hostile_AA_Ftr_03/02          3888  338.83
1.26    45.00   99.75   93.74
       293.99  Red_AG_Ftr_01/03        AA_Ftr_05/01                  3484  154.05
308.40   70.00   92.50   92.50
       294.00  Hostile_SAM_Killer_01   Asset_Def_SAM_03             28714  352.81
1.86    80.00   99.20   99.20
       294.75  Red_CM_01               Escort_01/01                 14202  277.92
315.80   66.58   99.73   99.73
       294.98  AA_Ftr_03/02            Hostile_AA_Ftr_04            27740  120.67
7.39    40.35   67.19   45.08
       295.35  Red_CM_01               Escort_01/02                 14467  277.79
316.27   65.14   99.91   99.91
       296.63  AA_Ftr_04/01            Hostile_AA_Ftr_03/03          2385  309.85
337.79   45.00   99.86   95.16
       296.63  AA_Ftr_04/01            Hostile_AA_Ftr_03/04          2385  309.85
337.79   45.00   99.92   96.26
       302.31  Hostile_AA_Ftr_03/01    AA_Ftr_04/02                  9587  250.55
359.61   70.00   91.00   78.91
       306.72  Red_CM_03               Escort_02/02                 15207  247.93
319.24   70.00   91.00   91.00
       310.89  AA_Ftr_04/01            Hostile_AA_Ftr_05             9931  186.14
0.28    45.00   99.96   96.26
       311.38  AA_Ftr_05/02            Hostile_AA_Ftr_07             6982   52.03
357.49   45.00   97.23   95.04
       311.48  Red_CM_03               Escort_02/01                 16297  247.88
322.68   70.00   97.30   97.30
       318.42  Hostile_SAM_Killer_01   Asset_Def_SAM_03             28538  346.97
1.93    80.00   99.84   99.84
       319.42  AA_Ftr_04/02            Hostile_AA_Ftr_05             9892  152.86
359.57   45.00   94.97   57.64
       322.90  Hostile_AA_Ftr_04       AA_Ftr_03/01                  1457  355.87
0.64    70.00   99.96   99.70
       329.11  AA_Ftr_04/02            Hostile_AA_Ftr_03/02          3623   49.02
359.63   45.00   97.23   63.36
       330.06  AA_Ftr_04/01            Hostile_AA_Ftr_05             5795  164.24
1.68    45.00   99.98   96.27
       338.26  AA_Ftr_04/02            Hostile_AA_Ftr_05             4373  190.15
0.10    45.00   98.48   63.43
       338.50  AA_Ftr_03/02            Hostile_AA_Ftr_05             8290  202.18
16.57   45.00   81.96   45.19
```

```
!
!                                PAPA Report
!                                -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!   Time              Target                   Engager                   Range      Az
El      SSPK    PKCum   Route
! (Seconds)            ID                       ID                       (m)      (Deg)
(Deg)                   PK
!------------  ------------------------- ------------------------- --------- ------- ----
--- ------- ------- -------
       340.82  Hostile_AA_Ftr_02/03    Asset_Def_SAM_03             52575   29.79
2.23   80.00   98.82   98.82
       350.18  AA_Ftr_04/02            Hostile_AA_Ftr_03/01          3968  211.66
0.88   45.00   99.16   66.90
       359.93  Hostile_AA_Ftr_07       AA_Ftr_05/02                 12332  192.95
4.72   64.01   99.76   96.81
       360.14  AA_Ftr_04/01            Hostile_AA_Ftr_03/01          5420  110.99
0.05   45.00   99.99   96.62
       362.09  Hostile_AA_Ftr_04       AA_Ftr_03/01                  4967  353.33
0.29   70.00   99.99   99.76
       367.31  Hostile_AA_Ftr_05       Escort_01/02                 13602   25.90
313.33   61.11   99.98   99.82
       371.78  Hostile_AA_Ftr_05       Escort_01/01                 13686   14.01
314.37   64.90   99.99   99.94
       372.00  Hostile_AA_Ftr_04       AA_Ftr_03/01                  1977   28.19
358.57   70.00  100.00   99.81
       378.44  AA_Ftr_04/02            Hostile_AA_Ftr_03/01          7720  346.73
359.79   45.00   99.54   70.04
       379.60  Hostile_AA_Ftr_04       AA_Ftr_03/01                  1280   63.87
356.09   70.00  100.00   99.84
       382.38  Hostile_AA_Ftr_05       Escort_02/01                 14228   66.07
316.50   62.95  100.00   99.98
       388.24  Hostile_AA_Ftr_03/01    AA_Ftr_04/01                  4554  277.89
359.95   70.00   97.30   79.41
       393.26  Hostile_AA_Ftr_06       Escort_01/02                 11944  354.46
307.09   70.00   99.20   98.55
       397.38  Hostile_AA_Ftr_03/02    AA_Ftr_04/01                  1854  223.45
0.58   70.00   91.00   70.71
       401.65  Escort_01/01            Hostile_AA_Ftr_07             8298  284.20
46.50   45.00   45.00   45.00
       402.26  Red_AG_Ftr_02/04        Escort_01/01                  9201  273.30
310.45   70.00   91.00   81.55
       406.38  Red_AG_Ftr_01/04        Escort_02/01                 10347  184.87
292.47   70.00   98.67   98.67
       409.42  Hostile_AA_Ftr_03/03    AA_Ftr_04/01                   756  109.20
355.58   70.00   84.87   50.77
       411.43  Red_AG_Ftr_02/03        Escort_02/01                  9308  160.77
298.53   70.00   97.46   97.46
       413.60  Escort_02/01            Hostile_AA_Ftr_03/04          9365  163.89
58.35   45.00   45.00   45.00
       415.29  AA_Ftr_04/01            Hostile_AA_Ftr_03/01          2155   46.88
5.83   45.00   99.99   96.94
       415.61  Hostile_AA_Ftr_07       Escort_01/02                  6139  155.58
295.28   70.00   99.93   99.04
```

```
!
!                                PAPA Report
!                                -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!   Time              Target                    Engager                Range      Az
El     SSPK    PKCum   Route
! (Seconds)             ID                        ID                    (m)      (Deg)
(Deg)                   PK
!------------  ------------------------ ------------------------ --------- ------- ----
--- ------- ------- -------
       422.51  Hostile_AA_Ftr_03/03    Asset_Def_SAM_03             55904   50.35
0.09    80.00   96.97   90.15
       423.97  AA_Ftr_03/01            Hostile_AA_Ftr_04             5616   78.68
359.97   45.00   98.48   72.20
       424.88  Red_AG_Ftr_02/03        Escort_01/02                  7819  219.78
317.39   70.00   99.24   99.24
       425.36  Hostile_AA_Ftr_03/02    Asset_Def_SAM_03             56485   49.43
0.09    80.00   98.20   94.14
       427.24  Hostile_AA_Ftr_03/04    Asset_Def_SAM_03             54597   49.05
0.13    80.00   89.92   89.92
       428.23  AA_Ftr_03/02            Hostile_AA_Ftr_04             5267   81.71
0.07    45.00   90.08   45.23
       430.00  Hostile_AA_Ftr_03/04    Escort_02/02                  8387   19.07
329.52   70.00   96.97   96.97
       434.25  AA_Ftr_03/01            Hostile_AA_Ftr_04             2746   52.39
357.92   45.00   99.16   72.22
       435.44  Red_AG_Ftr_02/04        Escort_01/02                  7995  262.00
341.80   70.00   97.30   94.46
       437.46  Red_AG_Ftr_02/04        Escort_02/02                  4917  178.37
307.41   70.00   99.19   98.34
       441.65  Hostile_AA_Ftr_03/03    AA_Ftr_04/02                 14709  147.83
359.43   63.06   98.88   92.01
       441.91  Escort_02/01            Hostile_AA_Ftr_03/01          3486  216.29
8.36    45.00   69.75   50.10
       445.00  Red_AG_Ftr_02/02        AA_Ftr_05/02                 18772  224.91
358.90   57.62   57.62   57.62
       448.24  Hostile_AA_Ftr_04       AA_Ftr_03/02                   741  333.65
17.65   70.00  100.00   99.90
       453.18  Hostile_SAM_Killer_01   Asset_Def_SAM_03             58948  356.91
0.70    80.00   99.97   99.97
       453.29  Missile 28769           Area_Defense_SAM_01         100625   12.64
79.24   63.96   63.96   63.96
       453.95  Missile 28770           Area_Defense_SAM_01         101048   12.99
79.61   64.91   64.91   64.91
       455.00  Red_AG_Ftr_02/03        Escort_02/02                  8024  153.03
331.60   70.00   99.77   99.77
       455.20  Red_AG_Ftr_01/03        Asset_Def_SAM_03             39174   61.23
0.42    80.00   98.50   98.50
       458.46  AA_Ftr_04/01            Hostile_AA_Ftr_03/01          4842   97.24
2.07    45.00  100.00   97.22
       460.30  Red_AG_Ftr_02/03        Escort_01/01                  4187  206.65
357.52   70.00   99.93   99.86
       463.00  Red_AG_Ftr_02/01        Escort_01/01                  2554  241.98
358.79   70.00   70.00   70.00
       465.01  AA_Ftr_04/02            Hostile_AA_Ftr_03/03          4498  305.52
359.80   45.00   99.75   71.12
       468.56  Escort_02/02            Hostile_AA_Ftr_03/04          9297  200.91
8.69    45.00   45.00   45.00
```

```
!
!                           PAPA Report
!                           -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!   Time              Target                     Engager                Range      Az
El      SSPK   PKCum   Route
! (Seconds)            ID                        ID                      (m)      (Deg)
(Deg)                   PK
!------------  ------------------------ ------------------------ --------- ------- ----
--- ------- ------- -------
        469.40  Red_AG_Ftr_01/02        Escort_01/01                 3176  226.40
0.06    70.00   70.00   70.00
        471.98  Missile 28771           Area_Defense_SAM_01        107808   32.84
67.79    55.01   55.01   55.01
        472.59  Missile 28772           Area_Defense_SAM_01        108038   33.25
68.13    56.04   56.04   56.04
        473.87  Red_AG_Ftr_01/01        Escort_01/01                 3079  247.68
359.33   70.00   85.00   69.25
        475.63  Red_AG_Ftr_02/01        AA_Ftr_05/02                16967  232.11
357.97   64.09   89.23   70.95
        477.42  Hostile_AA_Ftr_03/02    AA_Ftr_04/02                11151  149.66
359.89   70.00   99.46   95.33
        478.22  AA_Ftr_04/02            Hostile_AA_Ftr_03/03         1848  316.02
359.09   45.00   99.86   72.16
        480.96  Escort_02/02            Hostile_AA_Ftr_03/01        10180  260.68
1.25    45.00   69.75   50.10
        481.19  Escort_02/01            Hostile_AA_Ftr_03/04         5239  306.67
359.81   45.00   83.36   50.78
        482.24  Red_AG_Ftr_02/01        Escort_01/02                 4426  193.21
351.58   70.00   96.77   91.29
        492.14  Missile 28769           Asset_Def_SAM_01            28747   30.86
76.86    50.22   82.06   82.06
        493.04  Missile 28770           Asset_Def_SAM_01            29287   31.34
77.28    58.39   85.40   85.40
        493.24  Red_AG_Ftr_01/02        Escort_01/02                 2857  212.12
348.23   70.00   91.00   91.00
        493.58  Missile 28769           Asset_Def_SAM_01            28747   30.86
76.86    69.00   94.44   94.44
        494.50  Missile 28770           Asset_Def_SAM_01            29287   31.34
77.28    77.33   96.69   96.69
        499.18  Hostile_AA_Ftr_06      Point_Def_SAM_01/02           3044   25.54   25.01
50.00    99.60   99.27
        502.22  Escort_02/02            Hostile_AA_Ftr_03/04         5973  253.56
0.18    45.00   83.36   50.78
        502.30  Escort_02/01            Hostile_AA_Ftr_03/03        16203  251.22
359.89   45.00   90.85   52.54
        502.78  AA_Ftr_04/02            Hostile_AA_Ftr_03/02         4483  277.24
0.36    45.00   99.92   72.74
         504.57  Missile 28769         Asset_Def_SAM_01              7921   40.39   52.20
80.00    98.89    98.89
        504.67  Hostile_AA_Ftr_05       Point_Def_SAM_01/01          3588  290.57
14.92    50.00  100.00   99.99
        506.00  Missile 28770           Asset_Def_SAM_01             8242   40.36
53.83    80.00   99.34   99.34
        510.07  Missile 28771           Area_Defense_SAM_01         46746   38.80
38.51    70.00   86.50   86.50
        510.68  Missile 28772           Area_Defense_SAM_01         46974   38.90
39.05    70.00   86.81   86.81
```

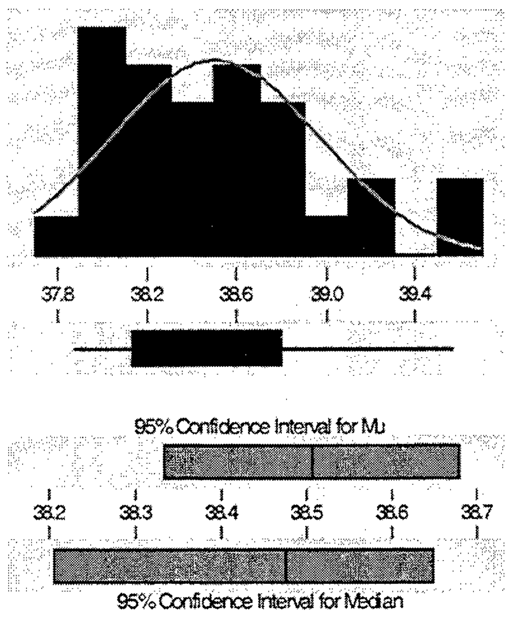
```
!
!                                        PAPA Report
!                                        -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!   Time              Target                     Engager               Range      Az
El     SSPK   PKCum   Route
! (Seconds)            ID                        ID                     (m)      (Deg)
(Deg)                   PK
!------------  ------------------------- ------------------------- --------- ------- ----
--- ------- ------- -------
       510.87  AA_Ftr_04/02          Hostile_AA_Ftr_03/02          1584  252.14    0.53
45.00   99.96   73.32
       511.20  Escort_02/01          Hostile_AA_Ftr_03/04          4296  289.22  359.86
45.00   94.97   53.19
       513.97  Hostile_AA_Ftr_06     Point_Def_SAM_01/02           7364  350.80   11.00
34.93   99.74   99.53
       517.42  AA_Ftr_04/02          Hostile_AA_Ftr_03/02           576  125.57    3.61
45.00   99.98   73.88
       522.31  Escort_02/02          Hostile_AA_Ftr_03/04          4713  320.59  359.76
45.00   90.85   51.45
       525.44  AA_Ftr_04/01          Hostile_AA_Ftr_03/03          2449  126.92   10.29
45.00  100.00   97.32
       533.88  AA_Ftr_04/01          Hostile_AA_Ftr_03/03          1700  132.01   20.23
45.00  100.00   97.42
       552.38  AA_Ftr_04/02          Hostile_AA_Ftr_03/02          3656  321.05  358.82
45.00   99.99   74.43
       556.23  Hostile_AA_Ftr_03/04  Escort_02/01                 12539  107.59    0.23
70.00   99.09   97.97
       564.37  Missile 28791         Area_Defense_SAM_01         102045   11.98   77.55
62.05   62.05   62.05
       565.36  Missile 28790         Area_Defense_SAM_01         109681    3.61   66.20
53.11   53.11   53.11
       565.66  AA_Ftr_04/02          Hostile_AA_Ftr_03/02          3437  325.46  358.76
45.00   99.99   74.97
       570.06  AA_Ftr_04/01          Hostile_AA_Ftr_03/03          3652  343.47  357.82
45.00  100.00   97.51
       597.53  Missile 28791         Area_Defense_SAM_01          43960   13.81   67.76
70.00   88.62   88.62
       603.50  Missile 28790         Area_Defense_SAM_01          49318    2.78   37.03
70.00   85.93   85.93
       605.72  Hostile_AA_Ftr_03/02  AA_Ftr_04/02                  8802  148.87  359.79
70.00   99.84   96.15
       617.64  Hostile_AA_Ftr_03/04  AA_Ftr_04/02                  3305   91.28    8.55
70.00   99.73   98.32
       618.01  Missile 28791         Asset_Def_SAM_01              8940   25.60   19.72
80.00   97.72   97.72
       630.44  Hostile_AA_Ftr_03/03  AA_Ftr_04/02                  2640   72.03   11.51
70.00   99.66   93.41
       728.48  Red_AG_Ftr_01/03      Asset_Def_SAM_03              5578  357.02    1.55
80.00   99.70   99.70
       730.89  Red_AG_Ftr_01/04      Asset_Def_SAM_03              6161    0.73    0.67
80.00   99.73   99.73
       733.69  Hostile_AA_Ftr_03/03  Point_Def_SAM_01/02           4502  356.60   12.61
50.00   99.83   96.71
       734.12  Red_AG_Ftr_02/03      Asset_Def_SAM_03              6740  359.01    0.87
80.00   99.99   99.97
       736.53  Red_AG_Ftr_02/04      Asset_Def_SAM_03              7354    2.12    0.50
80.00   99.84   99.67
```

```
!
!                                  PAPA Report
!                                  -----------
!
!      Scenario:  Demo300     Report generated on Thu May  6 07:43:10 1999
!
!      Report Type:  PK HISTORY
!
!      PAPA PK Legend
!      --------------
!       SSPK       Single Shot PK
!       PKCum      Cumulative PK Of Individual Shots
!       Route PK   Total PK For Killing Target
!
!    Time                 Target                    Engager                  Range      Az
El      SSPK   PKCum   Route
! (Seconds)               ID                        ID                        (m)      (Deg)
(Deg)                       PK
!------------  ------------------------- ------------------------- --------- ------- ----
--- ------- ------- -------
            738.94  Red_AG_Ftr_01/01      Asset_Def_SAM_03               7903  353.06    0.41
80.00   97.00   93.85
            741.38  Hostile_AA_Ftr_03/01  Point_Def_SAM_01/02            7329    8.35    9.40
35.47   98.26   86.71
            741.48  Red_AG_Ftr_02/01      Asset_Def_SAM_03               8577  353.73    0.36
80.00   99.35   98.26
            749.27  Red_AG_Ftr_01/03      Asset_Def_SAM_03              10175    2.50    0.40
80.00   99.94   99.94
            752.45  Red_AG_Ftr_02/03      Asset_Def_SAM_03              10795    4.11    0.04
80.00  100.00   99.99
            755.13  Red_AG_Ftr_02/04      Asset_Def_SAM_03              11490    8.15    0.12
80.00   99.97   99.93
            760.71  Red_AG_Ftr_02/01      Asset_Def_SAM_03              12938  356.21    0.25
80.00   99.87   99.65
            763.65  Red_AG_Ftr_01/01      Asset_Def_SAM_03              13864  354.14    0.36
80.00   99.40   98.77
            767.08  Red_AG_Ftr_01/02      Asset_Def_SAM_03              14537  348.55    0.34
80.00   98.20   98.20
            769.41  Red_AG_Ftr_02/02      Asset_Def_SAM_03              15437  353.74    0.26
80.00   91.52   91.52
            782.46  Red_AG_Ftr_02/03      Asset_Def_SAM_03              18824   14.64    0.43
80.00  100.00  100.00
            791.40  Red_AG_Ftr_02/01      Asset_Def_SAM_03              21310  346.55    0.27
80.00   99.97   99.93
            800.27  Red_AG_Ftr_01/04      Asset_Def_SAM_03              20742    2.52    0.10
80.00   99.95   99.95
            800.30  Red_AG_Ftr_01/01      Asset_Def_SAM_03              24526  346.44  359.93
80.00   99.88   99.75
            800.47  Red_AG_Ftr_02/04      Asset_Def_SAM_03              15338   45.44    0.36
80.00   99.99   99.99
            806.14  Red_AG_Ftr_01/03      Asset_Def_SAM_03              26455  348.84    0.12
80.00   99.99   99.99
            819.71  Red_AG_Ftr_02/02      Asset_Def_SAM_03              30302    4.25    0.22
80.00   98.30   98.30
            829.20  Red_AG_Ftr_02/03      Asset_Def_SAM_03              27968  351.93    0.11
80.00  100.00  100.00
            866.89  Red_AG_Ftr_02/04      Asset_Def_SAM_03              22466   22.82    0.41
80.00  100.00  100.00
            883.28  Red_AG_Ftr_01/03      Asset_Def_SAM_03              43355    3.17    0.14
80.00  100.00  100.00
            884.89  Hostile_AA_Ftr_03/04  Point_Def_SAM_01/02            5519  323.68   12.12
50.00   99.86   99.16
            904.15  Hostile_AA_Ftr_03/02  Point_Def_SAM_01/02            5145  341.00   14.36
50.00   99.92   98.07
            927.38  Red_AG_Ftr_02/04      Asset_Def_SAM_03              39697    1.32    0.26
80.00  100.00  100.00
            953.26  Red_AG_Ftr_02/03      Asset_Def_SAM_03              52871  358.04  359.99
80.00  100.00  100.00
```

APPENDIX K. PSTAT OUTPUT: DETERMINISTIC TRIALS

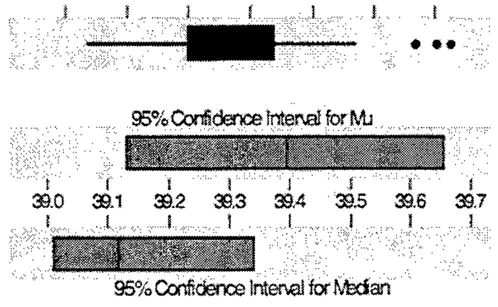
Note: Each page displays data gathered from unwrapped executables above data from wrapped executables (C3I, FP, Det).

Descriptive Statistics



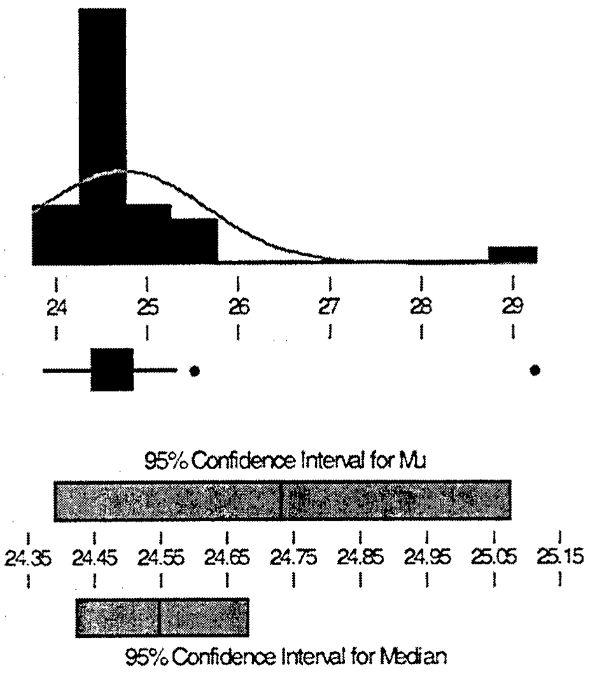
```
Variable: C3I
Cumulative CPU Times
(unwrapped c3i.pstat)
Anderson-Darling Normality Test
   A-Squared:          0.465
   P-Value:            0.236

   Mean              38.5063
   StDev              0.4658
   Variance         0.216996
   Skewness         0.609695
   Kurtosis         -5.1E-01
   N                      30

   Minimum           37.8800
   1st Quartile      38.1325
   Median            38.4750
   3rd Quartile      38.7900
   Maximum           39.5600
95% Confidence Interval for Mu
   38.3324           38.6803
95% Confidence Interval for Sigma
   0.3710             0.6262
95% Confidence Interval for Median
   38.2046           38.6477
```

Descriptive Statistics



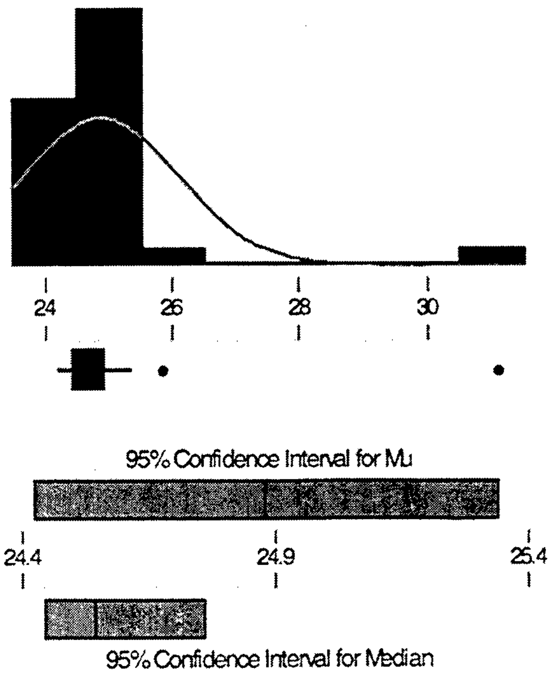
```
Variable: C3I
Cumulative CPU Time
(c3i.pstat)
Anderson-Darling Normality Test
   A-Squared:          2.236
   P-Value:            0.000

   Mean              39.3937
   StDev              0.7044
   Variance         0.496127
   Skewness          1.07294
   Kurtosis         0.315679
   N                      30

   Minimum           38.1800
   1st Quartile      38.9925
   Median            39.1150
   3rd Quartile      39.6700
   Maximum           41.1200
95% Confidence Interval for Mu
   39.1307           39.6567
95% Confidence Interval for Sigma
   0.5610             0.9469
95% Confidence Interval for Median
   39.0100           39.3400
```

Descriptive Statistics



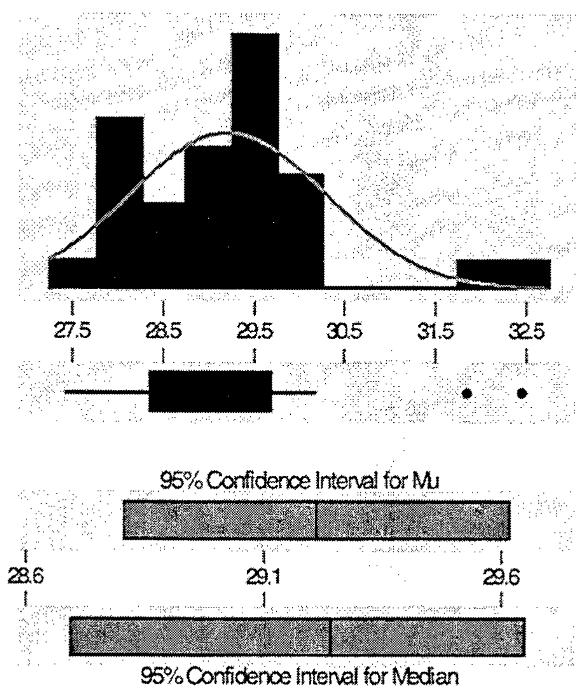
```
Variable: FP
Cumulative CPU Times
(unwrapped fp.pstat)
Anderson-Darling Normality Test
    A-Squared:        4.232
    P-Value:          0.000

    Mean            24.7327
    StDev            0.9188
    Variance       0.844179
    Skewness        3.87271
    Kurtosis        16.2088
    N                    30

    Minimum         23.8600
    1st Quartile    24.3925
    Median          24.5450
    3rd Quartile    24.8150
    Maximum         29.2300
  95% Confidence Interval for Mu
    24.3896         25.0757
 95% Confidence Interval for Sigma
    0.7317           1.2351
 95% Confidence Interval for Median
    24.4200         24.6809
```

Descriptive Statistics



```
Variable: FP
Cumulative CPU Time
(fp.pstat)
Anderson-Darling Normality Test
    A-Squared:        5.549
    P-Value:          0.000

    Mean            24.8813
    StDev            1.2281
    Variance        1.50816
    Skewness        4.29112
    Kurtosis        18.8038
    N                    30

    Minimum         24.2200
    1st Quartile    24.4250
    Median          24.5400
    3rd Quartile    24.8925
    Maximum         31.0900
  95% Confidence Interval for Mu
    24.4228         25.3399
 95% Confidence Interval for Sigma
    0.9780           1.6509
 95% Confidence Interval for Median
    24.4423         24.7577
```

Descriptive Statistics



Variable: Detect Cumulative CPU Times (unwrapped det.pstat)

Anderson-Darling Normality Test

A-Squared:	0.925
P-Value:	0.016
Mean	29.2117
StDev	1.0877
Variance	1.18299
Skewness	0.943513
Kurtosis	1.33429
N	30
Minimum	27.4300
1st Quartile	28.3475
Median	29.2400
3rd Quartile	29.6775
Maximum	32.4400

95% Confidence Interval for Mu

28.8055 29.6178

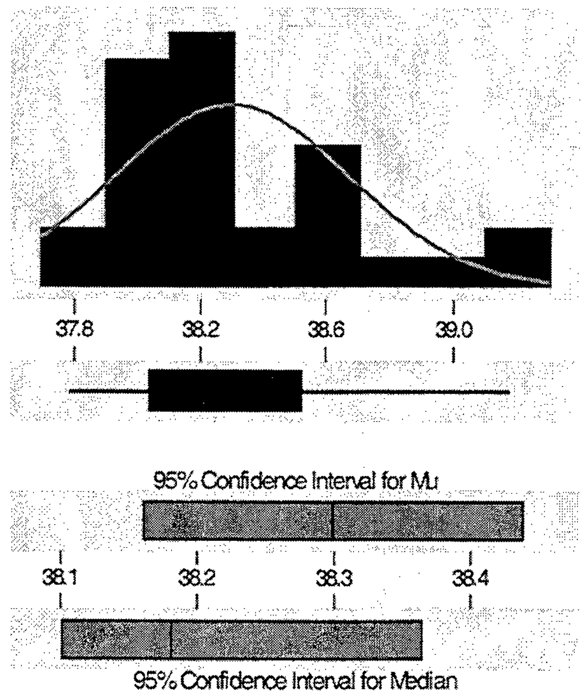
95% Confidence Interval for Sigma

0.8662 1.4622

95% Confidence Interval for Median

28.6926 29.6471

Descriptive Statistics



Variable: Detect Cumulative CPU Time (detect.pstat)

Anderson-Darling Normality Test

A-Squared:	1.014
P-Value:	0.010
Mean	38.2990
StDev	0.3719
Variance	0.138306
Skewness	0.907432
Kurtosis	-1.1E-01
N	30
Minimum	37.7900
1st Quartile	38.0375
Median	38.1800
3rd Quartile	38.5175
Maximum	39.1700

95% Confidence Interval for Mu

38.1601 38.4379

95% Confidence Interval for Sigma

0.2962 0.4999

95% Confidence Interval for Median

38.1014 38.3631

APPENDIX L: ACRONYMS

AE	Application Emulator
AOR	Area of Responsibility
ATO	Air Tasking Order
BMDO	Ballistic Missile Defense Organization
C2	Command and Control
C3	Command, Control, and Communications
C3ISIM	Command, Control, Communications, and Intelligence Simulation
C4I	Command, Control, Communications, Computers, and Intelligence
CL	Client Library
COMPASS	Common Operational Modeling, Planning, and Simulation Strategy
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency
DCA	Defensive Counter-Air
DCP	Distributed Collaborative Planning Tools
DII COE	Defense Information Infrastructure Common Operating Environment
DoD	Department of Defense
EADSIM	Extended Air Defense Simulation (formerly C3ISIM)
EEL	Executable Editing Library
ETC	Expected Time to Compute
FEZ	Fighter Engagement Zone
FISC	Flexible Integrated System Capability
FP	Flight Processing
GB	Gigabyte
GCCS	Global Command and Control System
GOTS	Government Off the Shelf
GUI	Graphic User Interface
HUMINT	Human Intelligence
I/O	Input and/or Output
IMINT	Imagery Intelligence
IPC	Inter Process Communication
IR	Infrared
M&S	Modeling and Simulation
MB	Megabyte
Mb/s	Megabit per Second
MD	MSHN Daemon
MEZ	Missile Engagement Zone
MOE	Measure of Effectiveness
MHz	Megahertz
MIPS	Millions of Instructions Per Second
MRS	Multiple Rocket System
MSHN	Management System for Heterogeneous Networks
NCA	National Command Authority
NFS	Network File System
NFS	Network File Service
NAS	Numerical Aerodynamic Simulation
OMA	Object Management Group
ORB	Object Request Broker
Pk	Probability of Kill
PSI-ID	Protocol Stack Instance Identifier
PSP	Protocol System Protocol
QoS	Quality of Service
RAM	Random Access Memory

ROM	Read Only Memory
RMS	Resource Management System
RPC	Remote Procedure Call
RRD	Resource Requirements Database
RSS	Resource Status Server
SA	Scheduling Advisor
SAM	Surface to Air Missile
SEAD	Suppression of Enemy Air Defense
SIGINT	Signals Intelligence
SMDC	Space and Missile Defense Command
SPAWAR	Space and Naval Warfare
TBE	Teledyne Brown Engineering
TBM	Tactical Ballistic Missile
TMD	Theater Missile Defense

LIST OF REFERENCES

[ARMS97] Robert Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1997.

[ANTH98] Robert W. Anthony, GCCS Evolution: Past, Present, and Future, *Proceedings of the 1998 Command and Control Research and Technology Symposium,* Naval Postgraduate School, Monterey, California, 1998.

[BERK91] Berkeley Unix Distribution, *Unix Man Pages*, March 1991.

[BRUN97] Matthias Brune, Jorn Gehring, and Alexander Reinefeld, Heterogeneous Message Passing and a Link to Resource Management, *Journal of Supercomputing, Vol 11, 1-17, 1997.*

[CASE91] Fred Case, Christopher Hines, and Steven Satchwell, *Analysis of Air Operations During Desert Shield / Desert Storm,* U.S. Air Force Studies and Analyses Agency, 1991.

[CARF99] Paul Carff, *Analysis on Resource Usage Information Granularity Required for Optimal Scheduling*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1999.

[CEBR98] VADM Arthur K. Cebrowski, USN, John J. Garstka, *Network-Centric Warfare: Its Origin and Future,* Naval Institute Proceedings, Naval Institute Press, January 1998.

[CHEN00] Cheng Heng Nom, *Testing of Various Instrumentation Resource Monitoring Tools on Win32/Intel Platform for MSHN's Applications,* Master's Thesis, Naval Postgraduate School, Monterey, California, expected March 2000.

[COMP99] Commander William Schlichter, *Common Operational Modeling, Planning and Simulation (COMPASS) Project History,* http://skyraider.spawar.navy.mil, January 1999.

[COOK99] Thomas S. Cook, *Dynamically Determining Distribution Statistics for Resources in a Distributed Environment*, Master's Thesis, Naval Postgraduate School, Monterey, California, expected September 1999.

[DESI98] Dr. Roberto Desimone, Andrew Preece, and Simon Hall, Improving Command Decision Making through the Integration of Joint Planning Aids into C2 Systems, *Proceedings of the 1998 Command and Control Research and Technology Symposium,* Naval Postgraduate School, Monterey, California, 1998.

[DOLG99] Max Dolgicer, CORBA and JAVA: Marriage or Just Serious Dating?, *Application Development Trends Magazine, Vol. 6, Number I,* January 1999.

[DRAK99] Tim Drake, *Distributed Real Time Application Emulator,* Master's Thesis, Naval Postgraduate School, Monterey, California, expected June 1999.

[DUMA98] Alpay Duman, *Design, The Use and Run-time Overhead of CORBA in MSHN Project,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1998.

[EADS98] Teledyne Brown Engineering, EADSIM Version 7.0, August 1998.

[FREU98] R. F. Freund and others, Scheduling Resources in Multi-user, Heterogeneous, Computing Environments with SmartNet, *Proceedings Eighth Heterogeneous Computing Workshop,* IEEE Computer Society, Los Alamitos, California, March 1998.

[HAYD97] Robbert van Renesse, Ken Birman, Mark Hayden, Alexey Vaysburd, and David Karr, *Building Adaptive Systems Using Ensemble,* Cornell University Technical Report, TR97-1638, July 1997.

[HENS99] Debra A. Hensgen and others, An Overview of MSHN: The Management System for Heterogeneous Networks, *Proceedings Eighth Heterogeneous Computing Workshop(HCW '99),* IEEE Computer Society, Los Alamitos, California, 1999.

[JAIN91] Raj Jain, *The Art of Computer Systems Performance Analysis,* John Wiley and Sons, Inc., New York, 1991.

[JOIN91] Joint Chiefs of Staff, *Joint Warfare of the US Armed Forces,* Washington, D.C., November 1991.

[JONG99] Jong-Kook Kim and others, *Priorities, Deadlines, Versions, and Security in a Performance Measure Framework for Distributed Heterogensous Networks.* In preparation for submission (1999).

[JOVI95] Joint Chiefs of Staff, *Joint Vision 2010,* Washington, D.C., 1995.

[KELT91] Avrill M. Law, W. David Kelton, *Simulation Modeling and Analysis,* 2d Edition, McGraw-Hill, New York, 1991.

[KIDD96] Taylor Kidd, Debbie Hensgen, Richard Freund, and Lantz Moore, "SmartNet: A Scheduling Framework for Heterogeneous Computing," *ISPAN*, 1996.

[KIDD98] Taylor Kidd, Debra Hensgen, Richard Freund, Matt Kussow, and Mark Campbell, *Compute Characteristics: A Useful Characterization of Job Run-times.* In preparation for submission (1998).

[KIDD99] Taylor Kidd and others, *Compute Characteristics*, Technical Report in progress, Naval PostGraduate School, Monterey, California, 1999.

[LARU95] James Larus and Eric Schnarr, *EEL: Machine-Independent Executable Editing*, SIGPLAN PLDI 95, 1995.

[LIVN95] Miron Livny, Michael Litzkow, Todd Tannenbaum, and Jim Basney, *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, Dr Dobbs Journal, February 1995.

[METH98] Teledyne Brown Engineering, EADSIM Version 7.0 *Methodology Manual*, August 1998.

[MCSW99] CDR Donald W. McSwain, *COMPASS and Collaborative Virtual Workspace* (A PowerPoint Brief) http://skyraider.spawar.navy.mil, February 1999.

[MOOR95] Lantz Moore and others, *Graze: A Tool for Performance Visualization and Analysis*, International Conference on Parallel Processing, 1995.

[MSMP95] Department of Defense Publication DoD 5000.59-P, *Modeling and Simulation Master Plan*, October 1995.

[NOBL97] Brian Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker, "Agile Application-Aware Adaptation for Mobility," *Proceedings of the 16th Symposium on Operating Systems*, 1997.

[NSTB96] National Research Council, *Computing and Communications in the Extreme,* National Academy Press, Washington, D. C., 1996.

[PEND93] William S. Pendergrass, *Combat Analysis for Command, Control, and Communications: A Primer, 1993 Edition,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1997.

[PUCA96A] Calton Pu, Tito Autrey, Andrew Black, Charles Consel, Crispin Cowan, Jon Inouye, Lakshmi Kethana, Jonathan Walpole, and Ke Zhang, *Optimistic Incremental Specialization: Streamlining a Commercial Operating System*, Oregon Graduate Institute, 1996.

[PUCA96B] Calton Pu, Tito Autrey, Jonathan Walpole, Crispin Cowan, and Charles Krasic, *Fast Concurrent Dynamic Linking for an Adaptive Operating System*, Oregon Graduate Institute, 1996.

[QUIN] Bob Quinn, Dave Shute, *Windows Sockets Network Programming,* Addison-Wesley, Reading Massachusetts, 1996.

[REED96] Daniel A. Reed and others, *I/O, Performance Analysis, and Performance Data Immersion,* Proceedings of MASCOTS96, 1996.

[REND97] Barry Render and Ralph M. Stair, Jr, *Qulaitative Analysis for Management,* 6th Edition, Prentice Hall, New Jersey, 1997.

[SCHM99] Douglas C. Schmidt, *Overview of CORBA,* http://www.cs.wustl.edu/~schmidt/corba-overview.html, February 1999.

[SCHN98] Matthew C. L. Schnaidt, *Design, Implementation, and Testing of MSHN's Application Resource Monitoring Library,* Master's Thesis, Naval Postgraduate School, Monterey, California, September 1997.

[SHAE00] Blanca Shaeffer, *Refining a Model for Use in Scheduling in MSHN,* Naval Postgraduate School, Master's Thesis, Monterey, California, expected March 2000.

[SING94] Mukesh Singhal, Niranjan G. Shivaratri, *Advanced Concepts in Operating Systems,* McGraw-Hill, Inc., New York, 1994.

[SKID99] Shirley Kidd and Matthew Schnaidt, *Tutorial on Wrapping Calls,* Naval Postgraduate School, 1999.

[SMDC97] US Army Space and Missile Defense Command, EADSIM Brochure and information packet, 1997, 1998.

[SNYD93] Frank M. Snyder, *Command and Control The Literature and Commentaries,* National Defense University, Washington, D.C., 1993.

[TBMC97] Lockheed Martin Command and Control Systems, *White Paper TBMCS Vision Today's Look at Tomorrow's System,* Colorado Springs, Colorado, September, 1997.

[TELE98] Teledyne Brown Engineering, EADSIM Training Course, October 1998.

[UTAR96] Laboratory for Parallel and Distributed Real-time Systems, *DeSiDeRaTa: Resource and QoS Management for Dynamic, Scalable, Dependable Real-Time Systems,* University of Texas at Arlington, 1996.

[VINO96] Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine, Vol. 35, No. 2,* February 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center...2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library...2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman, Code CS...1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93940-5000

4. Dr. Debra Hensgen...10
   Computer Science Department, Code CS/Hd
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Dr. William G. Kemple...1
   Command, Control, Communications Academic Group, Code CC/Ke
   Naval Postgraduate School
   Monterey, California 93943-5100

6. LCDR Wayne Porter...2
   910 N. Harbor Blvd. Apt 216
   La Habra, CA 90631

7. Mr. Brad Cooper...1
   Teledyne Brown Engineering
   300 Sparkman Drive (Mail Stop 105)
   Huntsville, AL 35805

8. Dr. Gary Koob...1
   DARPA / ITO
   3701 North Fairfax Drive
   Arlington, VA 22203-1714

9. Mr. Jim Watkins...1
   U.S. Army Space and Missile Defense Command
   P.O. Box 1500
   Huntsville, AL 35807-3801